Escuela Politécnica Superior

tapCuentas

Tu contabilidad más cerca que nunca

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor:

Andrea Ruiz Maciá

Tutor/es:

Otto Colomina Pardo

Universitat d'Alacant
Universidad de Alicante

Diciembre 2017

Dedicado especialmente a mis padres y a mi hermana, por llenarme estos años de apoyo y cariño.
Dedicado también a todos mis compañeros y profesores que han compartido conmigo estos años universitarios.

Contenido

1. Introducción

Esta memoria pertenece a un proyecto de final de carrera que, punto por punto irá describiendo el objetivo propuesto, siendo este el de la construcción de una **aplicación móvil multiplataforma** haciendo uso de las tecnologías más modernas y demandadas hoy en día.

Esta aplicación será, en resumen, una aplicación de **contabilidad doméstica** que permitirá al usuario introducir sus movimientos contables del día a día de forma fácil e intuitiva sin necesidad de vincular directamente a esta aplicación la cuenta bancaria del usuario.

El usuario podrá introducir movimientos de tipo ingreso, gasto, deuda o patrimonio y ver a tiempo real su resultado o situación financiera.

El enlace GitHub a mi proyecto, por si se quisiera examinar es: https://github.com/alirium/TFG---tapCuentas.git

1.1 Motivación personal

Cuando inicié mi camino hacia este proyecto, tenía claro en mi mente que como ingenieros informáticos que somos, estamos expuestos al gran avance que la tecnología alcanza día a día, es entonces cuando decidí que el punto fuerte de mi trabajo de fin de grado sería implementarlo con tecnologías que no se hubiesen utilizado anteriormente y que fueran punteras en el momento actual para así enfrentarme a la realización de un proyecto software por mí misma y ampliar mi conocimiento de cara al futuro.

Cuando finalizamos nuestro periodo de estudiante, entramos en el escenario del mundo laboral donde, especialmente en nuestro ámbito, se avanza a un ritmo alto y hay que estar actualizado si se quiere progresar. Es por todo esto que di comienzo a esta pequeña aventura que ha sido, **mi proyecto de final de carrera**.

Durante estos años en la Universidad hemos estado trabajando sobre todo en aplicaciones web, pero cada vez más los usuarios acceden a internet y a sus aplicaciones desde su dispositivo móvil, ya que es más accesible a lo largo del día. Por fortuna o no, los móviles se ha hecho una parte importante de nosotros ya que en él contenemos nuestra información, archivos, aplicaciones de uso diario, redes sociales... Ya no tenemos que esperar a tener disponible un ordenador para navegar por el mundo de la tecnología.

Teniendo en cuenta esto, decidí enfocar este trabajo a la plataforma móvil, que reúne los requisitos de estar en la vanguardia de nuevas metodologías y supone un reto de aprender tecnologías nuevas.

¿Cómo nace la idea de *tapCuentas*?

En mi casa se hace mucho uso de una aplicación de escritorio destinada a llevar un control de su contabilidad de una manera detallada y completa.

Es por ello por lo que me planteé el siguiente desafío: intentar desarrollar una aplicación móvil de contabilidad, pero sin llegar a tener la complejidad que puede llegar a embarcar este campo, de manera que fuera sencillo para cualquier usuario tanto entender el funcionamiento, como usarla.

Pero esto no era fácil, ya que no es lo mismo una aplicación de este tipo en un ordenador, donde puedes hacerla lo compleja e intuitiva que se proyecte debido a que, por una parte, las pantallas disponen de un mayor tamaño y, por otra parte, el usuario en un alto porcentaje de probabilidades estará tranquilo y centrado en lo que hace. En una plataforma móvil el usuario puede estar expuesto a muchas situaciones, puede estar en movimiento, con prisa… etc. Por lo que meter una aplicación de contabilidad tan completa como la que implementó mi padre, en un dispositivo móvil, no es lo óptimo.

Un concepto que aprendí en una de las asignaturas cursadas durante estos años es que, una aplicación móvil debe basarse en: una usabilidad sencilla, es decir, que el usuario sepa orientarse fácilmente sobre ella y en una complejidad de interfaz baja, cada '*tap'* o toque de pantalla cuenta, cuantos menos se necesiten para realizar una tarea específica, mejor.

Con ese concepto claro, y el previo aprendizaje sobre de contabilidad empieza el desarrollo de ***tapCuentas***.

2. Conceptos Previos

tapCuentas está basada en la contabilidad por lo que haremos una breve introducción a conceptos básicos de esta, que serán la base de la aplicación.

Estructura de una contabilidad simple de uso personal

Como en cualquier contabilidad la estructura se construye a partir de CUENTAS que el usuario crea y donde anota los movimientos que suponen un incremento o disminución de su solvencia.

En nuestro objetivo las cuentas se dividen inicialmente en dos Grupos:

Resultados

Las Cuentas de Resultados nos ofrecerán información sobre nuestra capacidad de generar ahorro o si por el contrario nuestra actividad está generando mayor endeudamiento o pérdidas.

Las Cuentas de Resultados se dividirán, a su vez, en otros dos Grupos:

En las Cuentas censadas como ingresos estarán todas aquellas que representan una generación de fondos, como, por ejemplo: Una Nómina, Intereses que nos paga el Banco, etc.

En las Cuentas censadas como gastos estarán todas aquellas que identifican en qué nos gastamos el dinero. Por ejemplo: Gastos del Coche, Gastos Electricidad o Gastos Personales.

Situación

Las Cuentas de Situación nos ofrecerán información sobre nuestra solvencia o Capital.
Las Cuentas de Situación se dividirán, a su vez, en otros dos Grupos:

Las cuentas censadas como patrimonio identificarán los bienes que tenemos. Ejemplos de títulos para una Cuenta de Patrimonio: Cuenta del Banco, Plan de Pensiones, Depósito de Acciones, Plazo Fijo en un Banco, Persona que nos debe dinero, Piso o Vivienda, Nuestro Coche, Nuestro Ajuar (ordenador, consola, muebles), etc.

En las cuentas censadas como deuda estarán aquellas que identifican deudas que tenemos adquiridas con terceros.
Ejemplos de títulos para una Cuenta de Deuda: Préstamo del Banco, Hipoteca, Anticipo de la Nómina, Persona a la que debemos dinero, etc.

En resumen, los Ingresos incrementan los Resultados, los Gastos reducen los Resultados, el Patrimonio incrementa la Situación y las Deudas disminuyen la Situación.

Entendida la naturaleza que debe tener cada cuenta tenemos que decidir entre dos tipos de proyecto de contabilidad. La podemos definir como **simplificada** o de **doble asiento**.

En esta primera versión de *tapCuentas* haremos uso de una naturaleza **simplificada**, es decir, en las cuentas no existirá una dependencia o correlación de unas con otras.

3. Estudio del arte

3.1 Producto y estudio de mercado

Ahora, hagamos un breve estudio sobre esta aplicación.
Actualmente el mundo de las aplicaciones móviles está muy desarrollado, cientos de aplicaciones móviles son subidas diariamente a los 'market'.
Esto hace difícil desarrollar una aplicación sin que exista ya otra con las mismas características o parecidas. Pero esto también puede enfocarse como un reto.

¿Qué puede ofrecer mi aplicación que las demás no?
¿Qué puedo mejorar de las demás?
¿Por qué utilizar la mía y no las otras?

Antes de empezar el proyecto, hice una pequeña investigación de las aplicaciones disponibles para móvil destinadas a gestionar la contabilidad. Como esperábamos, actualmente existen ya algunas en el mercado para llevar un control de gastos e ingresos.
Tras hacer una pequeña encuesta en mi circulo de conocidos y familiares, la mayoría de ellos afirmaron no confiar en introducir sus cuentas bancaras en aplicaciones de terceros por lo que mi aplicación de contabilidad descarta la sincronización con las cuentas bancarias personales. *tapCuentas* está más enfocada a llevar una contabilidad **personal y libre**.

Una vez descartadas aplicaciones con este tipo sincronización, me instalé en mi dispositivo personal algunas de las aplicaciones que se ajustaban más a la idea de tapCuentas para realizar un estudio.

Tres de ellas fueron: "Gastos", "Monedero definitivo" y "Gastos monedero".

Después de trabajar con ellas, pude sacar las siguientes conclusiones.
Por un lado, había aspectos que se podían mejorar; La primera, "Gastos" se limitaba a trabajar con movimientos de tipo ingreso y gasto, la segunda, "Modero definitivo" se limitaba únicamente a movimientos de tipo gasto" y por último la tercera, "Gastos monedero" tenía una interfaz muy poco intuitiva, tuve problemas para entender cómo trabajar con ella además de una interfaz un poco desfasada visualmente.

Por otro lado, también había aspectos positivos que me ayudaron a enfocar *tapCuentas*; "Gastos" y "Monedero definitivo" tenían una interfaz visualmente buena y la primera, además bastante intuitiva, la segunda, "Modero definitivo" y por último, "Gastos monedero" contaba un sistema de *login*.

Una vez finalizado este pequeño estudio, empecé a desarrollar mi idea. Mi aplicación de contabilidad personalizada se basaría en que un usuario pudiera hacer los cuatro movimientos básicos de la contabilidad de manera sencilla y con una interfaz bonita.

¿Pero cómo hacer esto de manera sencilla y rápida?

Tras muchos bocetos fallidos e intentos para lograr una mayor simplicidad de uso sin perder ninguno de los requerimientos principales, realicé mi esquema final que queda resumido en los siguientes puntos (los mockups estarán más adelante).

- Un usuario podrá realizar los movimientos:
  - Gasto.
  - Ingreso.
  - Patrimonio.
  - Deuda.
- Cuando se realice un movimiento se tendrá que seleccionar una cuenta existente o crear una nueva.
- Podrá ver los resultados.
- Podrá ver su situación.
- Tendrá visualización de datos en gráficas.

Ya tenemos las ideas claras sobre el funcionamiento, ahora ¿Qué tecnologías hemos usado? Pasemos al siguiente punto.

4. Tecnologías

Actualmente los usuarios de dispositivos móviles no usan un sistema operativo común. En el mercado destacan entre otros, dos grandes compañías, *Android* e *iOS*. Además, nuestra aplicación no requerirá una gran potencia, por lo que no es de extrañar que, de cara a una futura distribución, *tapCuentas* se decidiera desarrollar como aplicación multiplataforma.

Siendo de esta forma, quedan descartadas herramientas de desarrollo dedicadas a un sistema operativo en concreto como puede ser *Android Studio* que, aunque ofrece una óptima potencia desarrollando la aplicación como nativa Android, no es lo que buscamos en esa ocasión.

El criterio de elección de las tecnologías ha sido el de investigar y elegir las más nuevas, modernas y demandadas actualmente de cara a cumplir el objetivo de este TFG aprender las nuevas tecnologías por uno mismo y sirvan de cara al futuro profesional. Vamos a detallar por partes que tecnologías se han empleado y una descripción de ellas.

4.1 Front-end

Angular 2

Aprendiendo de las carencias de la primera versión del framework AngularJS, Angular 2 trae muchas mejoras que, junto a su actual popularidad, hizo que se eligiera.

Con angular se ha podido crear una aplicación multiplataforma pudiendo hacerla enfocada tanto a *Android* como a *iOS* en un mismo código.

Gracias a su estructura MVC (Modelo -vista- controlador) cada componente tiene asociado un archivo vista y controlador por lo que se hace mucho más dinámica la programación.

TypeScript

El lenguaje utilizado es TypeScript que es JavaScript, pero añadiendo tipado estático y objetos, lo que hace mucho más fácil trabajar con Angular 2 ya que su estructura es mucho más ordenada, tipada y comprensible que JavaScript, asimilándose de esta manera a lenguajes vistos en la carrera.

Angular CLI

Esta herramienta de línea de comandos ha ayudado a empezar a desarrollar rápidamente, añadir componentes y pre visualizar de forma instantánea la aplicación haciendo más ágil el desarrollo.

Ionic 2

Para el estilo y la personalización de la vista se ha utilizado este framework, ya que junto a Angular 2 es el más idóneo, documentado y popular de cara a aplicaciones móviles multiplataforma. En su página oficial encontramos muchos componentes y ejemplos que nos han servido en el desarrollo.

HTML

Cada componente tiene asociado un archivo HTML donde va el contenido de su vista, en este archivo se implantan los componentes Ionic.

CSS

Hemos personalizado clases de los elementos de la web con CSS en una clase general para todo el proyecto.

4.2 Back-end

Swagger

Swagger es una herramienta con posibilidad de uso en línea que ayuda a crear la estructura del CRUD de nuestra aplicación y su implementación en el lenguaje deseado.

Node.js

Como lenguaje para el back hemos utilizado node.js ya que es flexible y está a la orden del día disponiendo así de alta documentación.

4.3 BBDD

MongoDB

Últimamente se escucha hablar de MongoDB, una base de datos no SQL que guarda en colecciones los elementos de nuestra aplicación.

Nos olvidamos del concepto tablas y vemos que cada elemento se guarda como un array conteniendo todos sus datos, esto nos proporciona agilidad y facilidad a la hora de trabajar con datos.

Robo 3T

Es el programa de interfaz visual de las bases de datos MongoDB.

4.4 Despliegue

Docker

Docker te permite empaquetar una aplicación o un nodo de esta, haciéndola independiente del entorno de ejecución, es decir, permite la movilidad de la app ya sea Windows, Linux o plataformas "en la nube" como puede ser Amazon Web Services.

4.5 Entorno Desarrollo

Sistema Operativo

Hemos trabajado todo el rato bajo Windows 10 debido a que estamos más acostumbrados a él y es más fácil instalar diversas herramientas.

IDE

Hemos trabajado con *Atom* ya que es fluido y gracias a sus múltiples extensiones podemos instalarnos todo tipo de ayudas, como una consola asociada o un autocompletar.

La implementación en el proyecto de estas tecnologías será explicada con más detalle en el apartado 4.

Depuración

El navegador *Google Chrome* nos ha provisto de una simulación de la aplicación durante su desarrollo para ir viendo como quedaba a tiempo real en distintos dispositivos, ya fuera un Android, un iOS o un ordenador. Además, nos provee de una consola que es muy útil en la depuración de código JavaScript.

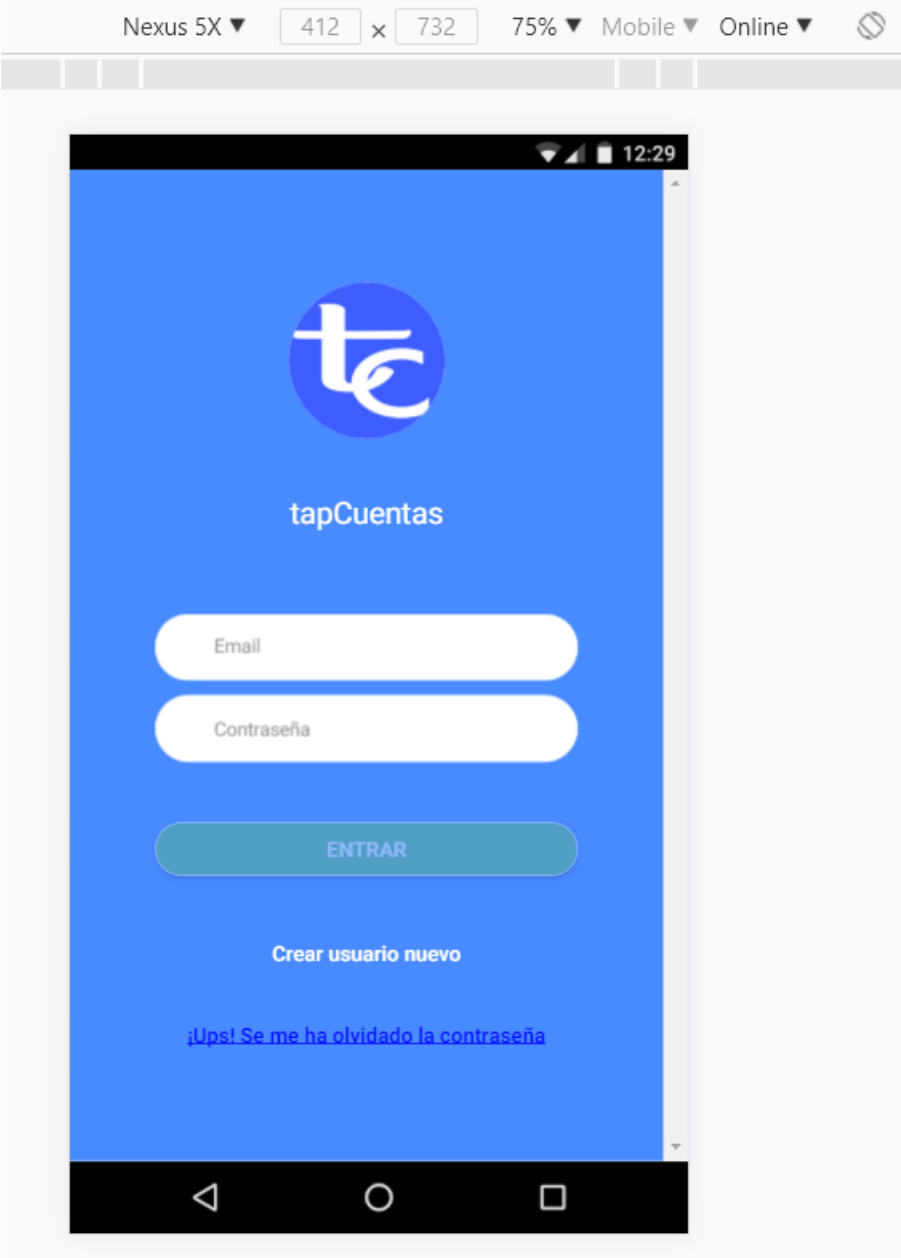

Ilustración 1 - Simulación tapCuentas en nexus5

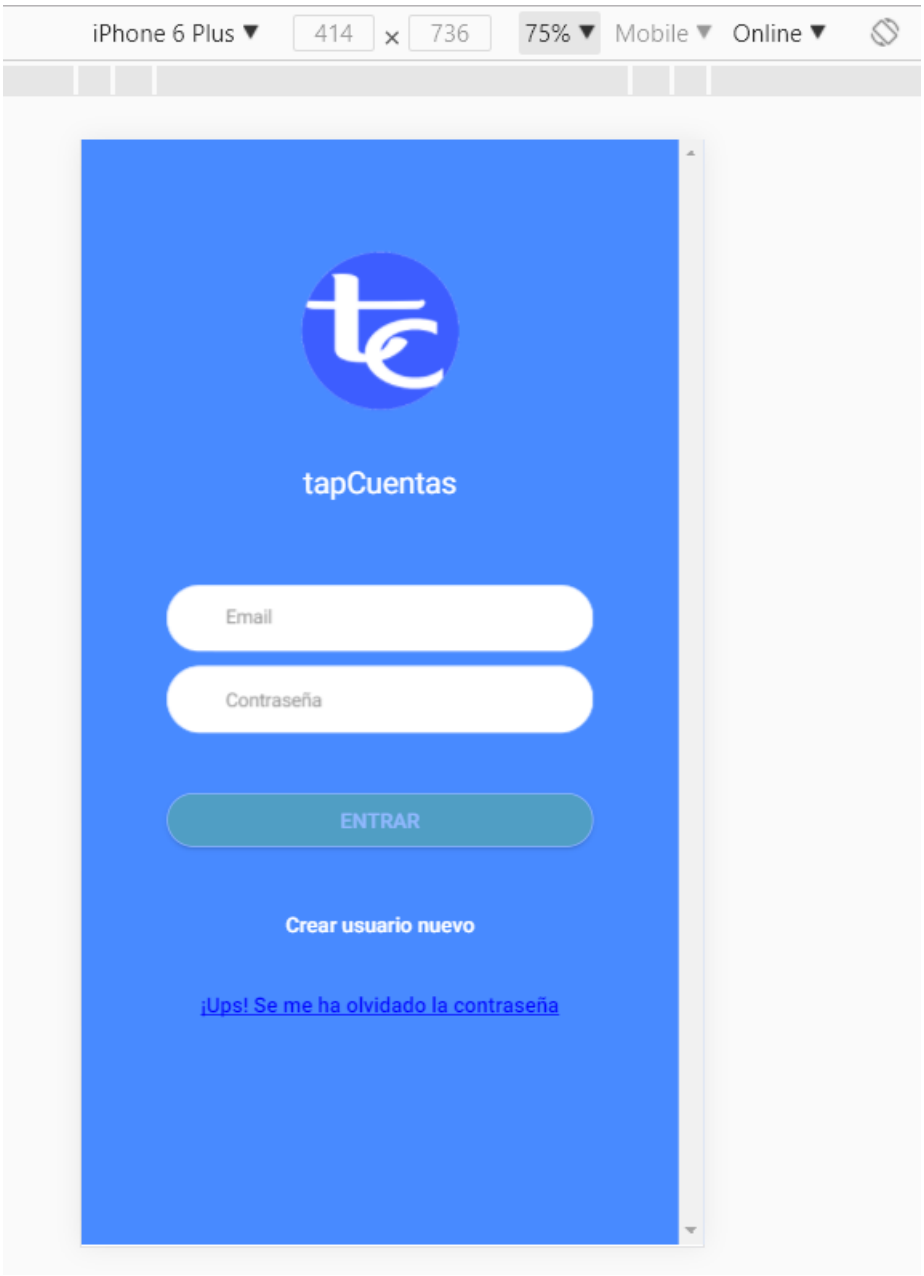

Ilustración 2 - Simulación tapCuentas en iPhone 6

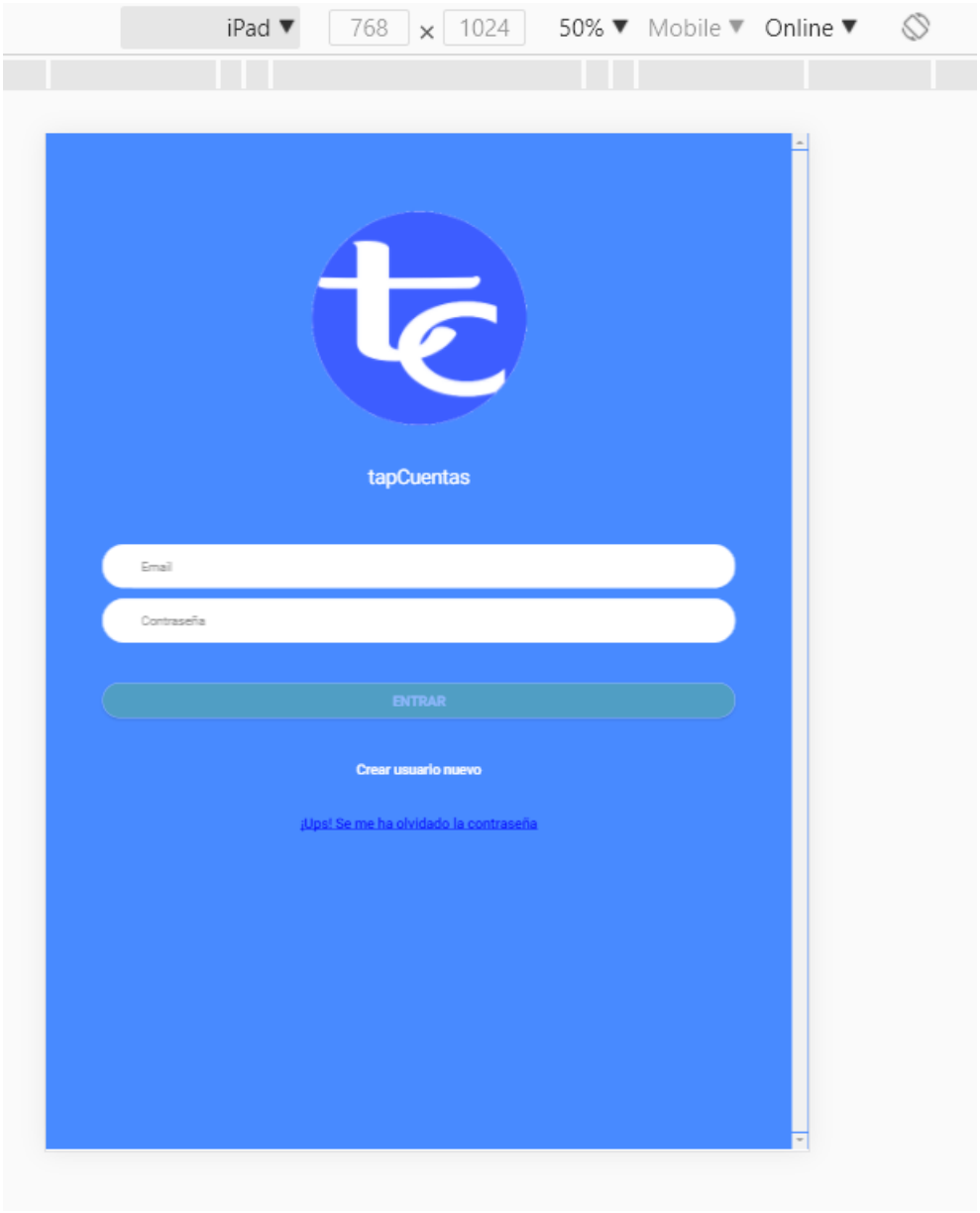

Ilustración 3 - Simulación tapCuentas en iPad

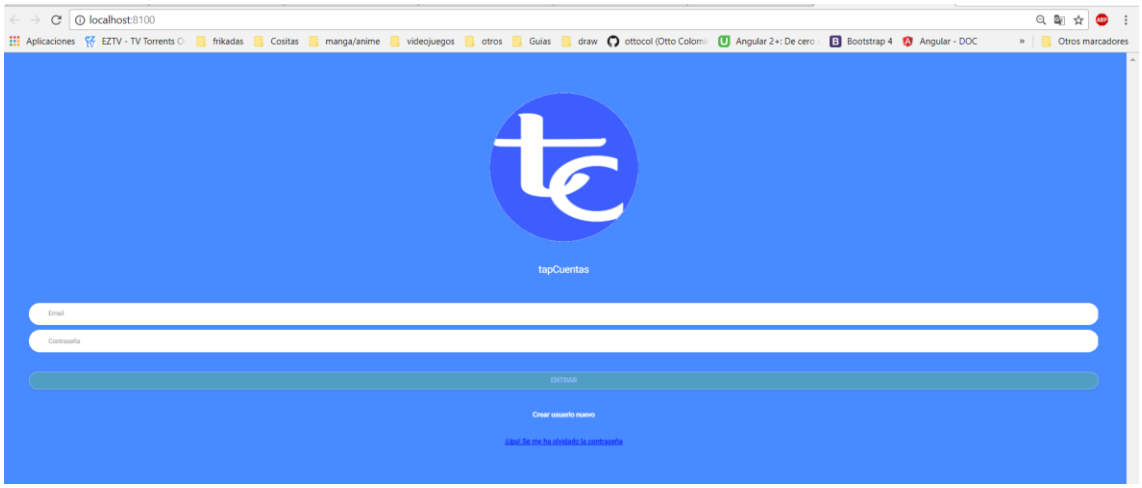

Ilustración 4 - Simulación tapCuentas en un ordenador

5. Metodología desarrollo

En este apartado se comentarán metodologías y técnicas empleadas en el desarrollo de esta aplicación.

5.1 Metodología Ágil

Las metodologías de desarrollo son necesarias en el proceso de creación de todo proyecto puesto que sin ellas la organización seria pobre y no existiría un control del trabajo en el tiempo, exponiéndonos así a caer en la comodidad de dejar el trabajo para el último momento dando lugar pérdida de funcionalidades y errores inesperados.

La planificación se ha dividido en metas que estuvieran divididas en periodos de tiempo cortos obteniendo así un resultado constante. Por lo tanto, se optó por una metodología de desarrollo ágil, haciendo así un procedimiento de trabajo iterativo e incremental que se adaptaba perfectamente a lo deseado.

No hemos implementado una metodología **Scrum** pura puesto que el equipo de trabajo, en este caso, no dispone de un gran número de integrantes, sino de una persona, por lo que la coordinación no era necesaria. Pero no se dudó en organizar el proyecto en *sprints* para llevar un control sobre trabajo y tiempo como hemos citado anteriormente.

Para llevar a cabo todo el proceso de construcción siguiendo estas pautas, se han elegido las siguientes herramientas descritas.

5.1.1 TRELLO

Trello es una herramienta de administración de proyectos que emplea el sistema *Kanban*. Este nos permite a través de una bonita interfaz dinámica, crearnos tarjetas con tareas definidas que pueden irse alternando en columnas de trabajo.

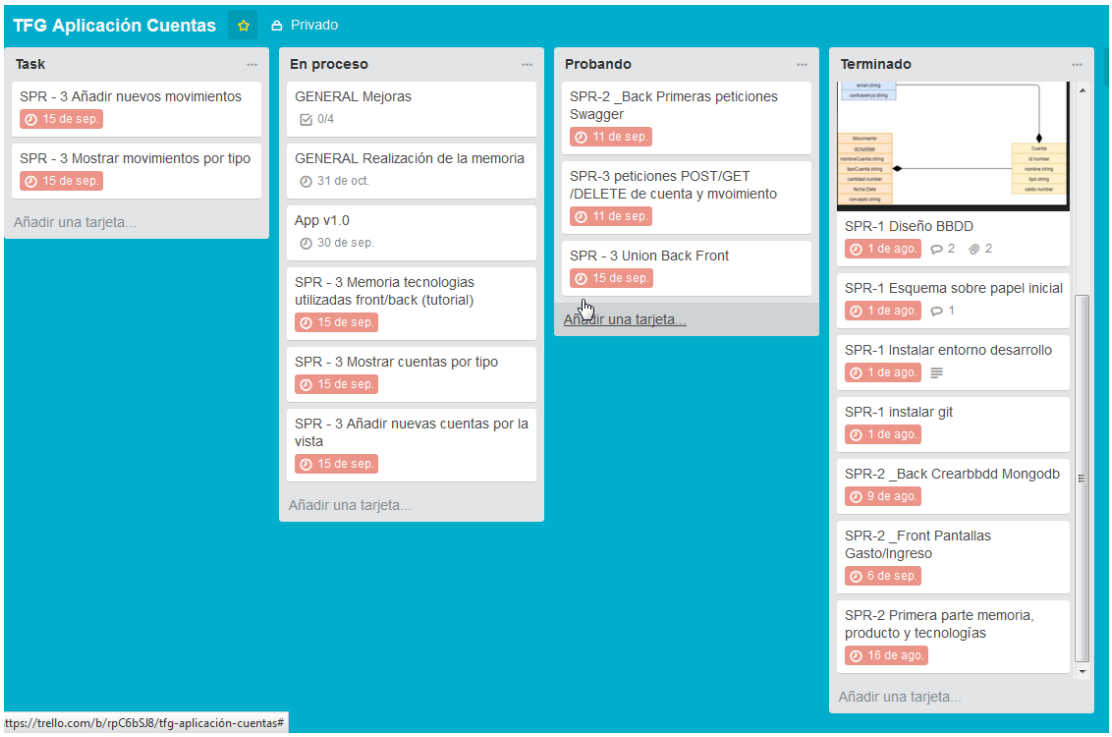

Ilustración 5 - Visualización del tablero TRELLO utilizado en el desarrollo

Como se puede observar en la imagen anterior, el escritorio de Trello se constituye de múltiples columnas que indican el estado de una tarea concreta; en proceso, probando, finalizada…

Las tareas que llevaremos a cabo en el proyecto están especificadas en paneles más pequeños a los que llamaremos, "tarjetas". En cualquier momento se puede ver el estado de desarrollo de una tarea.

Lo bueno de esta herramienta es que además de ser gratuita, puedes ir añadiendo en cualquier momento a cada tarjeta comentarios, listas, archivos adjuntos… Y compartir el tablero con otros usuarios si fuera el caso.

Las tareas se han dividido principalmente por bloques de desarrollo, primero las partes más generales y según avanzaba el desarrollo y se comenzaba el siguiente *sprint* las partes más específicas.

Al principio de cada sprint se han estructurado tarjetas añadiendo o modificando según fuera menester añadiendo siempre un identificador y una fecha de caducidad a cada tarea para poder llevar una mejor organización.

5.2 Control de versiones

5.2.1 GIT

Todo informático a lo largo de su carrera ha aprendido, a base de experiencia, que cuando se está desarrollando un proyecto siempre se debe crear regularmente, una copia de seguridad del trabajo realizado hasta el momento por si dado el momento se pierde o se hace un cambio que afecte gravemente al resto del desarrollo.
Por lo que es muy importante y podríamos decir necesario, llevar un control de versiones.

En este caso se ha optado por utilizar la herramienta git, utilizada frecuentemente en nuestros trabajos la universidad.
Con esto nos aseguramos tener una copia siempre de nuestro proyecto en la nube y un control sobre los cambios que se han realizado a lo largo del desarrollo.
No comentamos la posibilidad de multiusuario en el mismo proyecto puesto que no es el caso.

He utilizado la página de *GitHub* para tener acceso a todos los movimientos realizados en *git* y su contenido.

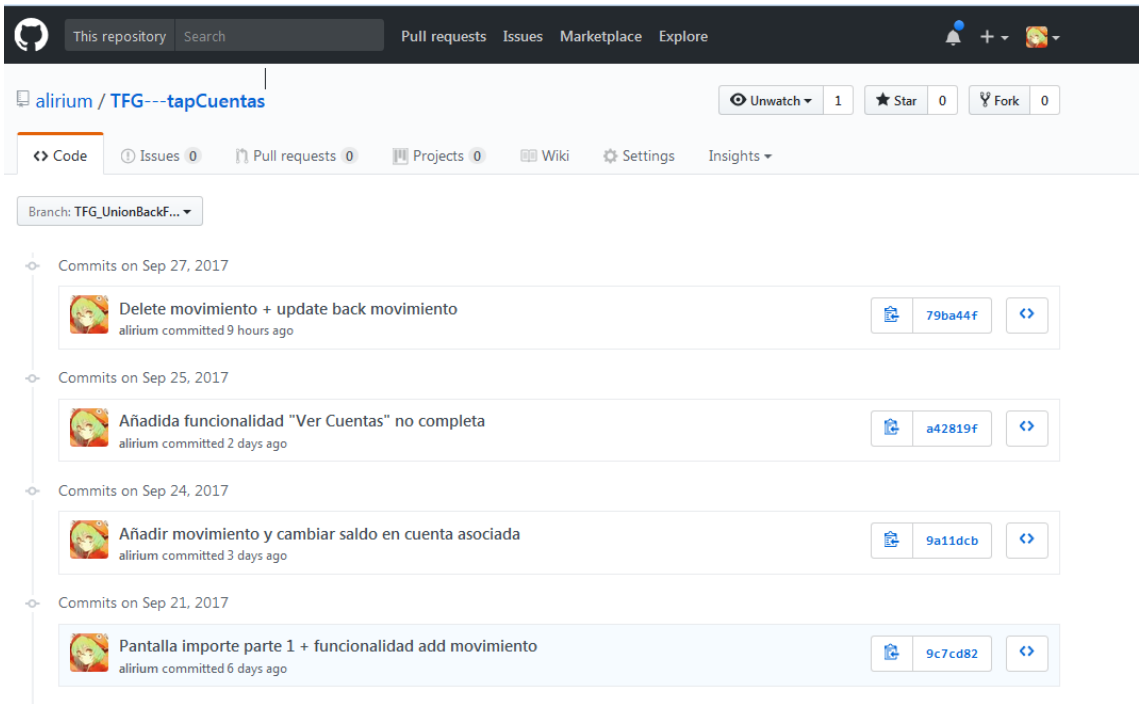

Ilustración 6 - Ejemplo de seguimiento de commits en GIT

El enlace de *GitHub* al proyecto es el siguiente, por si se desea examinar el código, o hacer uso de la aplicación: https://github.com/alirium/TFG---tapCuentas.git

5.2.2 SOURCETREE

Personalmente la interfaz que provee la consola para git no terminó de convencer puesto que la información que aparece es muy escasa, por lo que decidí hacer uso de SourceTree.

SourceTree es una herramienta que dota de una interfaz a los controles de versiones.

Es muy visual sobre el desarrollo; En todo momento muestra el estado de las ramas, las que hay creadas, las que tienes en local y en la que estás trabajando. Además, se pueden ver todos los commits que se han ido haciendo en cada una de ellas, quien es el autor y que archivos se han modificado.

Cuando se han hecho cambios simplemente haces 'click' en los botones de acción e integras los cambios.

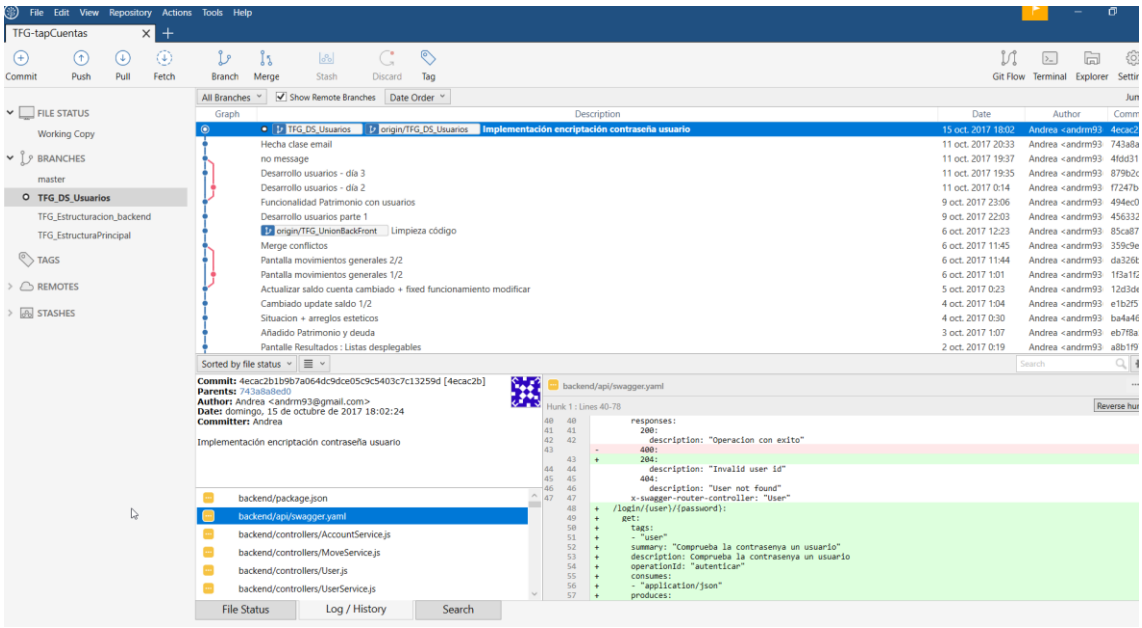

Ilustración 7 - Captura de una vista de trabajo del programa SourceTree

5.3 Interfaz visual

La parte visual de una aplicación es de las más importantes, ya que es con la que el usuario va a interactuar, por ello hay que realizar un esquema previo antes de ponerse a codificar.

Una vez tenemos una visión previa de la aplicación, la programaremos de una manera más ordenada y clara.

Hemos realizado varias versiones de diseño o mockups que se han ido mejorando con desarrollo.

5.3.1 HERRAMIENTAS

La aplicación ha contado con varias versiones en lo que el aspecto visual se refiere. Los mockups se han realizado tanto a mano como con software específico.

myBalsamiq

Este programa fue el elegido para hacer los primeros mockups ya que es muy fácil de usar. Aunque es de pago la universidad nos concedió una licencia de un par de meses para ayudar en la realización de estos. Con myBalsamiq podemos programar los mockups de forma que al pinchar en una parte específica nos lleve a otro mockup, simulando así una navegación

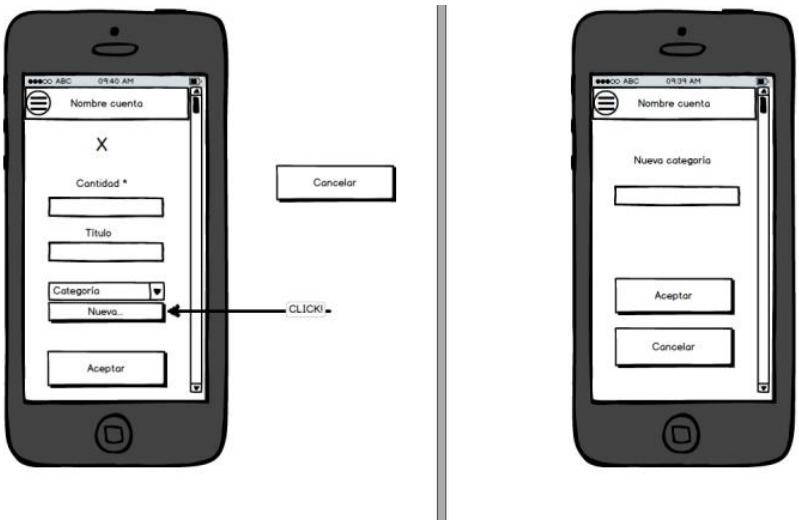

Ilustración 8 - Primer mockup pantalla "Nueva Cuenta"

5.3.2 DISEÑO

Se ha optado por una interfaz bonita y asociativa:

- La paleta de colores se compone de 6 colores
  - Verde: Todo lo relacionado con ingresos y operaciones agregar.
  - Rojo: Todo lo relacionado con gastos y operaciones de borrado.
  - Naranja: Todo lo relacionado con operaciones de patrimonio.
  - Rosa: Todo lo relacionado con operaciones de deuda.
  - Azul y blanco: Colores base.
- Una interfaz sencilla e intuitiva:
  - Botones para acciones principales.
  - Listas para selección de cuentas disponibles
  - Formularios sencillos.
  - Menú estilo *tabs* para una visión permanente de las opciones principales.
  - Deslizadores para navegar entre distintos tipos de cuentas en las acciones principales.

El icono elegido para la aplicación ha sido desarrollado personalmente con estética sencilla.

Ilustración 9 - Icono creado para tapCuentas

6. Documentación Técnica

En este apartado se detalla qué modelos y estructura se han seguido en el proyecto y la implementación de las tecnologías.

6.1 Arquitectura

A continuación, vemos un esquema general de la arquitectura implementada.

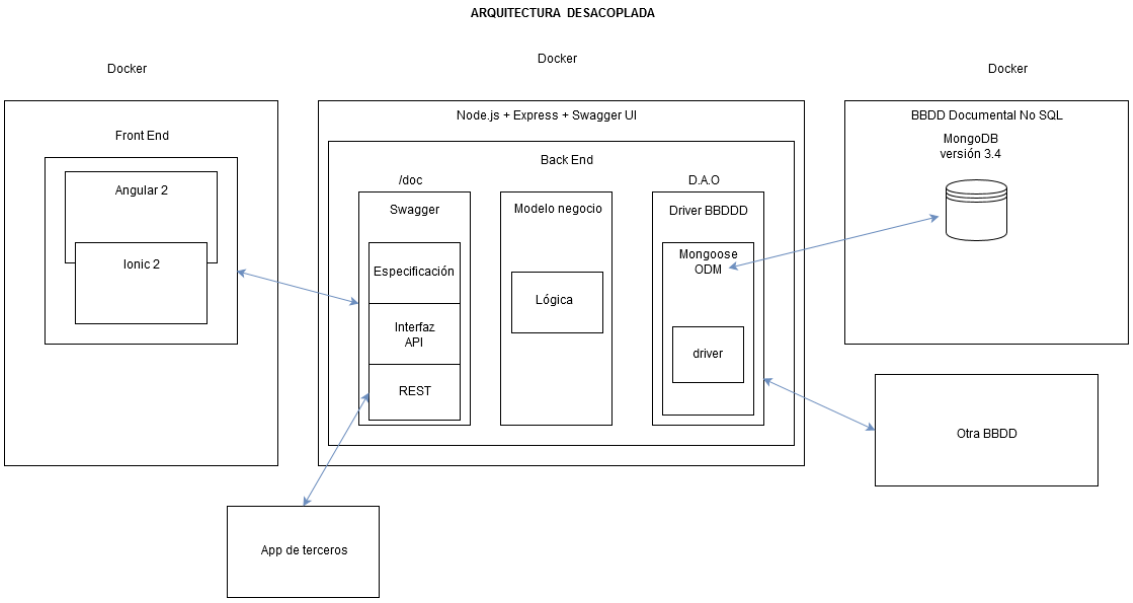

Ilustración 10 - Esquema general de la arquitectura

La arquitectura elegida es una arquitectura desacoplada donde cada una de las partes va a ser independiente de la otra, tenemos una arquitectura **cliente-servidor** con el código dividido en capas.

La base de datos va a encontrarse en el servidor, por lo que sea cual sea el dispositivo desde el que se conecte el usuario, podrá acceder a sus datos ya sea desde un móvil en la calle o desde una Tablet desde casa; Además así si tuviera problemas con su dispositivo habitual, no perdería sus datos.

Además la implementación de Docker facilitará el despliegue y puesta en producción de la app en cualquier plataforma

6.2 Base de datos

6.2.1 DESARROLLO

Como hemos citado anteriormente hemos utilizado una base de datos MongoDB. Con el inicio de este proyecto nos introducimos en las bases de datos No SQL como es el caso de MongoDB, siendo muy útil para bases de datos sencillas olvidándonos de implementar relaciones y dependencias directas con claves si no que se guardan como documentos. Su estructura es similar a la de un JSON.

Las entidades principales de este proyecto serán tres:

- Usuario
- Cuenta
- Movimiento

Cada cuenta y movimiento tendrán un campo *usuario* en el cual se guardará el **id** del usuario asociado, para que cuando este acceda a la aplicación, sólo tenga acceso a sus datos.

Lo mismo será con la relación de los movimientos con sus respectivas cuentas.

Vemos a continuación un esquema relacional tradicional para visualizar las relaciones como hemos aprendido en la carrera:

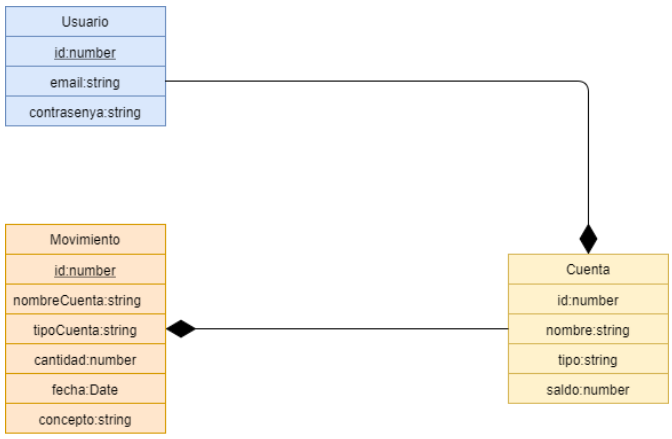

Ilustración 11 - Esquema relacional entidades

Trasladado a la visión de una base de datos documental MongoDB sería:

```
#Usuario
{"_id" : ObjectId(""),
 "email" : "email del usuario",
 "contrasenya" : "contrasenya del usuario"}

#Cuenta
{"_id" : ObjectId(""),
 "nombre" : "nombre de la cuenta",
 "tipo" : "tipo de la cuenta",
 "saldo" :  saldo,
 "usuario" :  "email del usuario asociado"}

#Movimiento
{"_id" : ObjectId(""),
 "nombreCuenta" : "nombre de la cuenta asociada",
 "tipoCuenta" : "tipo",
 "cantidad": cantidad,
 "fecha" : fecha,
 "concepto" : "concepto del movimiento",
 "usuario" :  "email del usuario asociado"}
```

Ilustración 12 - Esquema documental base de datos MongoDB

Para poder tener una visión y trabajar con los datos almacenados en todo momento hemos utilizado el programa Robo 3T.

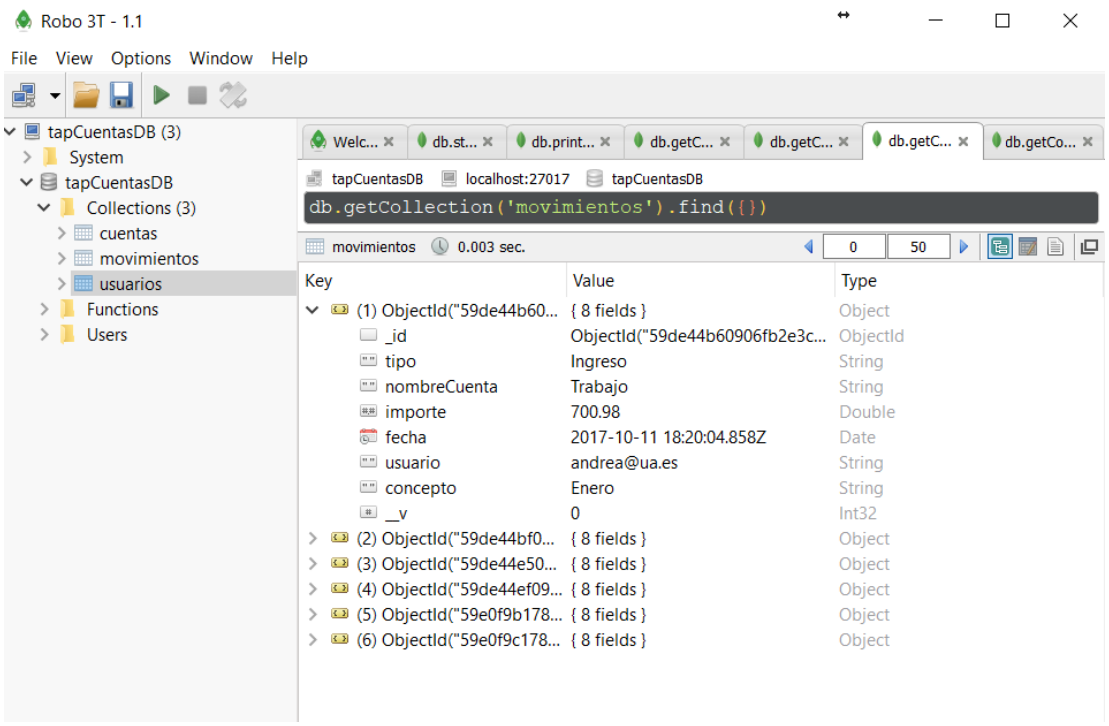

Ilustración 13 - Representación del esquema de un usuario en Robo 3T

Hemos creado la base de datos llamada **tapCuentasBD.** Para **modelar** las entidades hemos usado **mongoose**, lo que nos ha permitido crear desde el front las estructuras de los documentos que, más tarde se irán guardando aquí.

Ilustración 14 - Logo y eslogan de mongoose

El campo *contraseña* del usuario ha sido encriptado con ayuda de la librería **bcrypt.**

Ilustración 15 - Captura de la estructura de un usuario en la base de datos.

6.2.2 EJECUCIÓN

La base de datos se ejecuta de la siguiente manera en la carpeta que hemos seleccionado para guardar los datos:

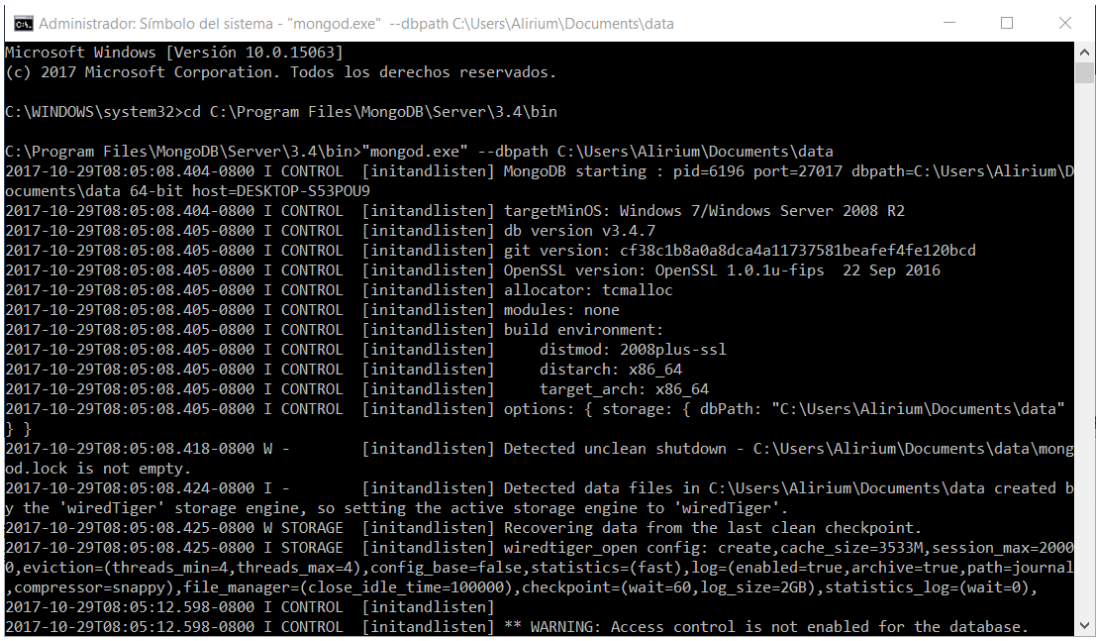

Ilustración 16 - Ejecución base de datos MongoDB

6.3 Back

Empecemos viendo un esquema general del back.

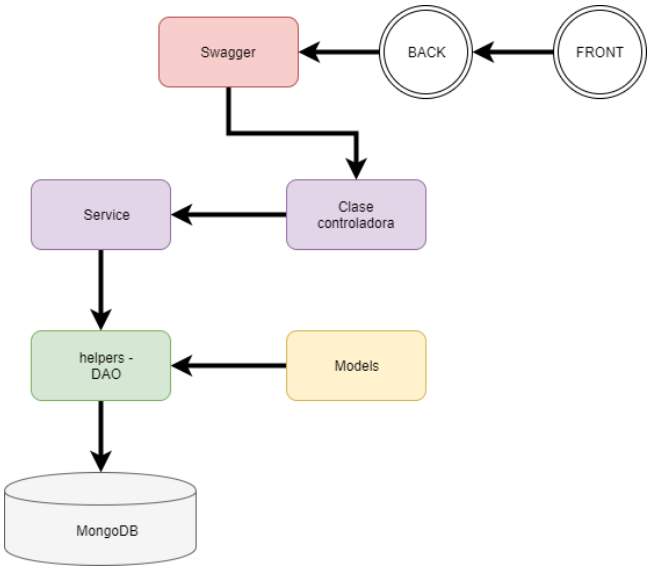

Ilustración 17 - Esquema funcionamiento planteado en el back

6.3.1 DESARROLLO

La estructura principal del back se comenzó con ayuda de la herramienta *Swagger* y su editor online. Esta herramienta funciona a tiempo real y te va avisando de los errores que puedes ir cometiendo.

En esta primera estructura pusimos los detalles de las principales operaciones **CRUD** de cada entidad de las que se iba a hacer uso y que se llamarán desde el **front**:

Ilustración 18 - Visualización de la creación del esqueleto CRUD de la aplicación con ayuda del editor online Swagger

Cuando se terminó esta fase, generamos automáticamente el código **servidor** en lenguaje *node.js*, lo que nos ahorró tiempo de codificación y facilitó la implementación.

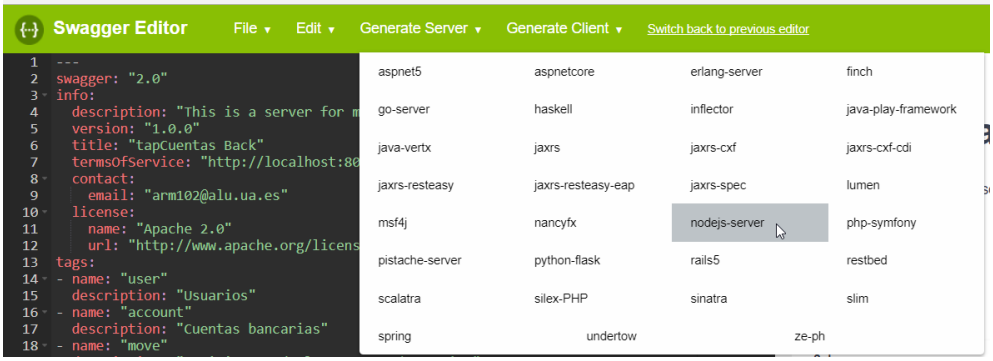

Ilustración 19 - Posibles lenguajes de servidor de los que dispone Swagger

Una vez tenemos parte del código generado ya podemos empezar a codificar nuestro servidor.

Este ha sido realizado en *node.js* por su potencia a la hora de gestionar peticiones http gracias a su manera asíncrona de trabajo, soporta multiplataforma y cuenta con una gran comunidad detrás.

A continuación, explicaremos la estructura u organización que ha sido implementada.

En nuestra carpeta principal del proyecto, contamos con dos subcarpetas, una que contiene el **back end** y otra el **front end.** En este apartado nos centramos en la de **back end.**

En esta contamos con varias subcarpetas para una mejor organización del código que es muy importante a la hora de más tarde trabajar con muchas clases que se relacionan entre sí.

/api: En esta carpeta guardamos el archivo anteriormente generado con Swagger que redirigirá las peticiones del **front** al **back**.

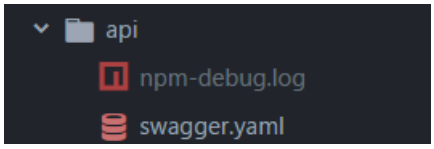

Ilustración 20 - Carpeta api del proyecto

/controllers: Aquí se guardan dos tipos de ficheros, los que agrupan las distintas funciones que tendrá cada entidad (Account.js, User.js…) y las que tienen el código que realizan cada una de esas funciones (AccountService.js, UserService.js...). Estos ficheros unen la estructura creada en Swagger con los **D.A.O** de cada entidad.

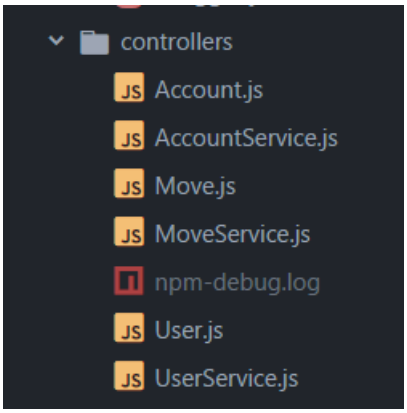

Ilustración 21 - Carpeta controllers del proyecto

```
'use strict';

var url = require('url');

var User = require('./UserService');

module.exports.deleteUser = function deleteUser (req, res, next) {
  User.deleteUser(req.swagger.params, res, next);
};

module.exports.addUser = function addUser (req, res, next) {
  User.addUser(req.swagger.params, res, next);
};
```

Ilustración 22 - Ejemplo de código de la clase User.js

```
/**
 * Obtiene todos los usuarios
 *
 **/
exports.getUsers = function(args, res, next) {
   UsuarioDAO.getAll(function(err, response) {
     if (err && response) {
       res.setHeader('Content-Type', 'application/json');
       res.statusCode = 404;
       res.end(JSON.stringify({msg: 'No existen usuarios.', response: response}));
       return;
     } if (err) {
       res.setHeader('Content-Type', 'application/json');
       res.statusCode = 400;
       res.end(JSON.stringify(err));
       return;
     } else {
       res.setHeader('Content-Type', 'application/json');
       res.statusCode = 201;
       res.end(JSON.stringify({msg: 'Se han encontrado usuarios.', response: response}));
       return;
     }
   });
}
```

Ilustración 23 - Ejemplo de la clase UserService.js

/helper: En esta carpeta se encuentran los archivos **D.A.O** de cada entidad, es decir las funciones de cada clase que conectan directamente la base de datos, en este caso MONGODB, con el código que recibe del back.

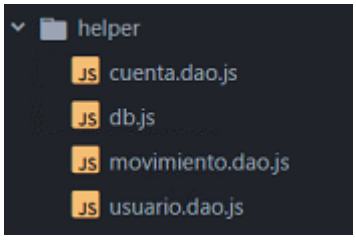

Ilustración 24 - Carpeta helper del proyecto

```
'use string'

var Cuenta = require('../models/Cuenta');
var MovimientoDAO = require('./movimiento.dao');

/*Actualizar una cuenta*/
function updateAccount(_id, _cuenta, callback){
  var myquery = {_id: _id };
    var newvalues = {
      nombreCuenta: _cuenta.nombreCuenta,
      usuario:  _cuenta.usuario
     }
  Cuenta.update(myquery, newvalues, function(err, res) {
   let modificados = res.nModified;
   if (modificados == 1 && (null, res)) {
       callback(null, modificados,"cuenta " + _id + " modificado");
    }
    else {
     callback("No se ha podido modificar", modificados, null);
    }
  });
}
```

Ilustración 25 - Ejemplo de clase Cuenta.dao

/models: En esta carpeta creamos la estructura de las entidades principales que serán guardadas en la base de datos creada en mongoDB.

Aquí hacemos uso del anterior mencionado **mongoose** para modelar los objetos.

Ilustración 26 - Carpeta models del proyecto

```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;
var UsuarioSchema = new Schema({
   nombreUsuario    : {type: String}
  , contrasenya     : {type: String}
});
module.exports = mongoose.model("Usuario", UsuarioSchema);
```

Ilustración 27 - Ejemplo de la clase Usuario

Otros archivos de la carpeta ***backend:*** En esta carpeta también disponemos de archivos de configuración.

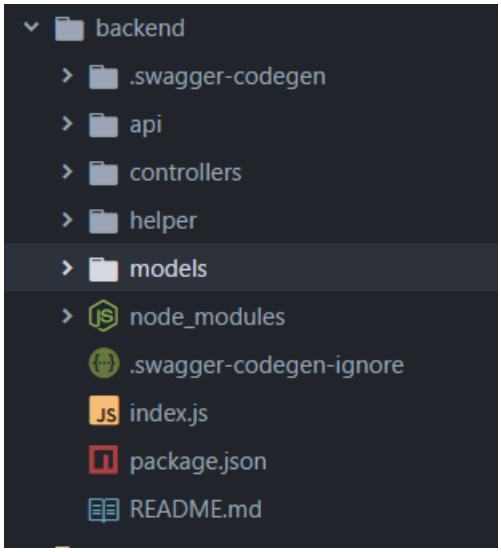

Ilustración 28 - Carpeta backend

Destacamos entre ellos el de **package.json** donde van incluidos todos los paquetes que hemos instalado durante la implementación del código necesarios para su funcionamiento y que será indispensable a la hora de ejecutar el servidor en cualquier máquina.

```
{
  "name": "swagger-tapCuentas",
  "version": "1.0.0",
  "description": "This is a sample server Petstore server
  "main": "index.js",
  "scripts": {
    "prestart": "npm install",
    "start": "node index.js"
  },
  "keywords": [
    "swagger"
  ],
  "license": "Unlicense",
  "private": true,
  "dependencies": {
    "bcrypt": "^1.0.3",
    "connect": "^3.2.0",
    "cors": "^2.8.4",
    "js-yaml": "^3.3.0",
    "mongoose": "^4.11.7",
    "swagger-tools": "0.10.1"
  }
}
```

Ilustración 29 - Clase package.json del proyecto

6.3.2 EJECUCIÓN

Para ejecutar la parte del back, primero arrancamos la base de datos mongoDB:

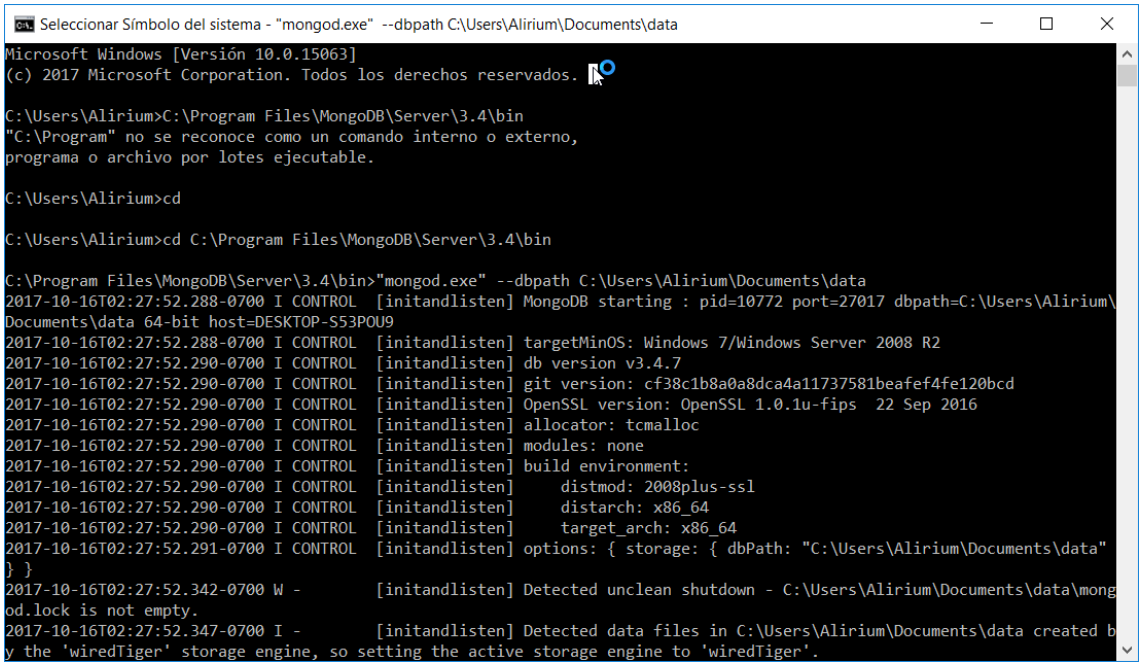

Ilustración 30 - Ejemplo ejecución base de datos MongoDB

Y luego la parte del servidor:

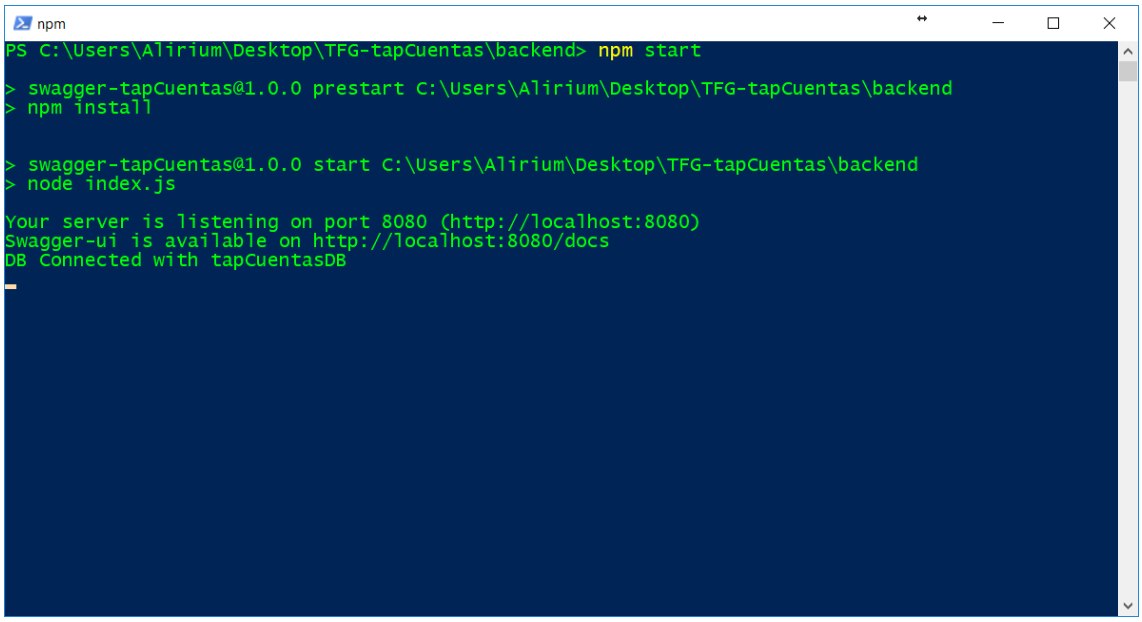

Ilustración 31 - Ejemplo ejecución del servidor node.js

Y ya tenemos el backend ejecutándose en el puerto 8080, si accedemos a él.

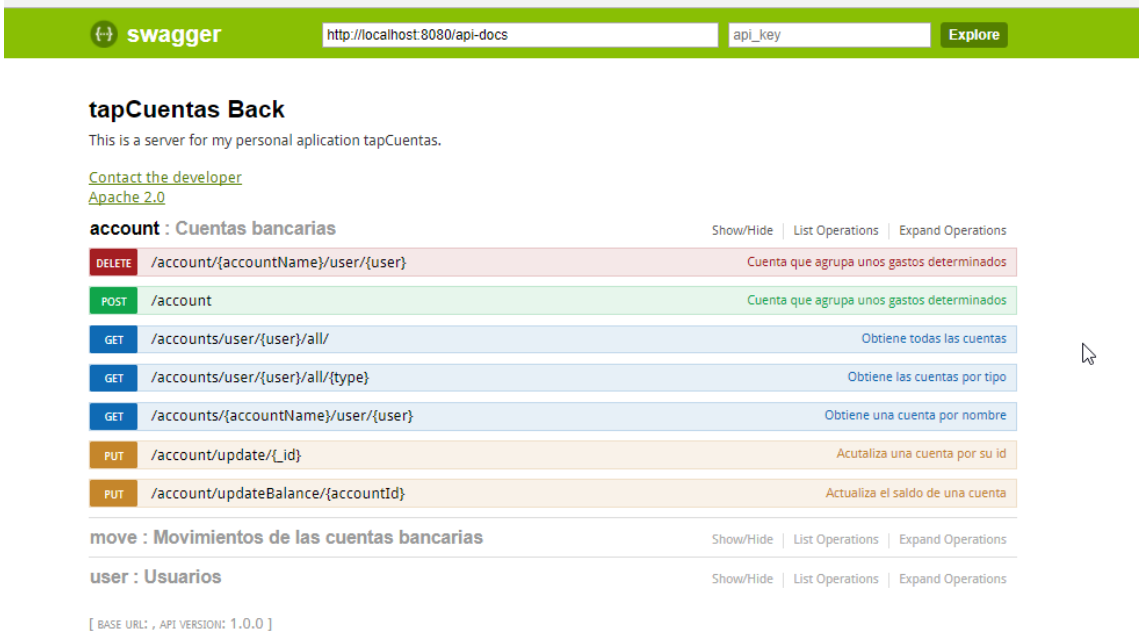

Ilustración 32 - http://localhost:8080/docs/#/

6.3.3 TESTING

Una vez arrancamos la base de datos y el servidor procedemos al ***testing***.

Se han implementado test automatizados con la herramienta de **postman** que hemos podido agrupar en colecciones personalizadas**.**

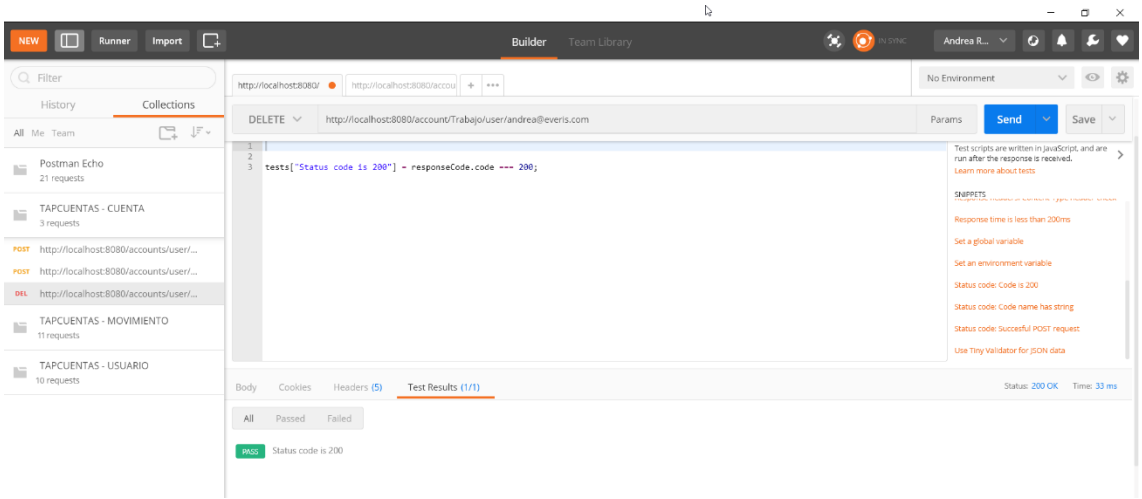

Ilustración 33 - Ejemplo test unitarios del back en postman

Las pruebas que no se han podido realizar con esta herramienta, las hemos realizado con la interfaz UI que Swagger proporciona, accediendo a la URL: http://localhost:8080/docs/#/account nos aparecerá la siguiente pantalla con la que podremos trabajar:

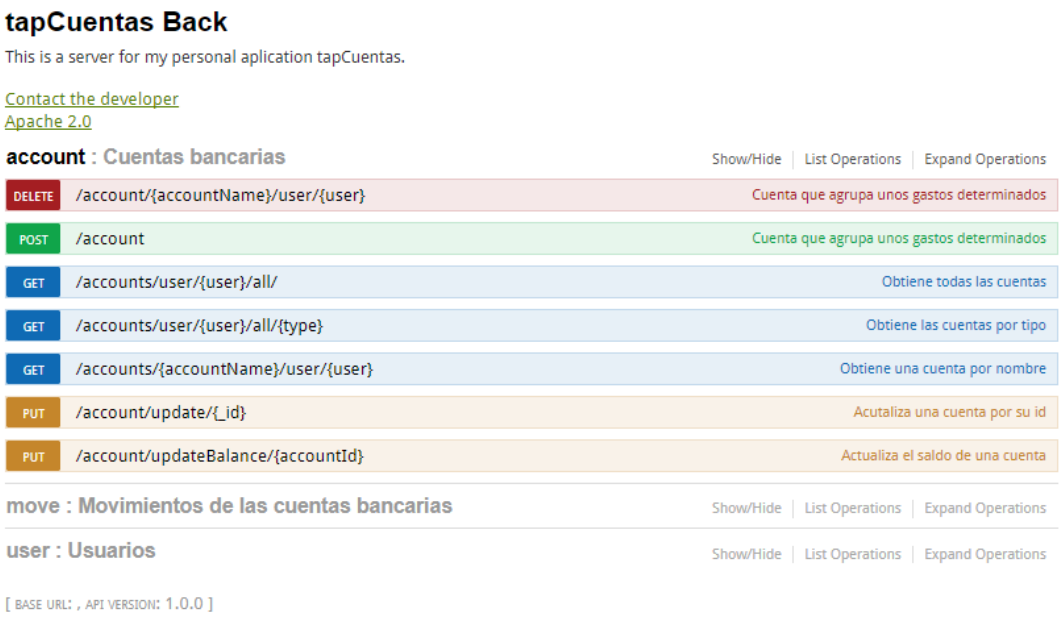

Ilustración 34 - Ejemplo de la interfaz de Swagger del back de la entidad Cuenta

Podremos seleccionar una operación CRUD que ha sido implementada en el back y probarla.

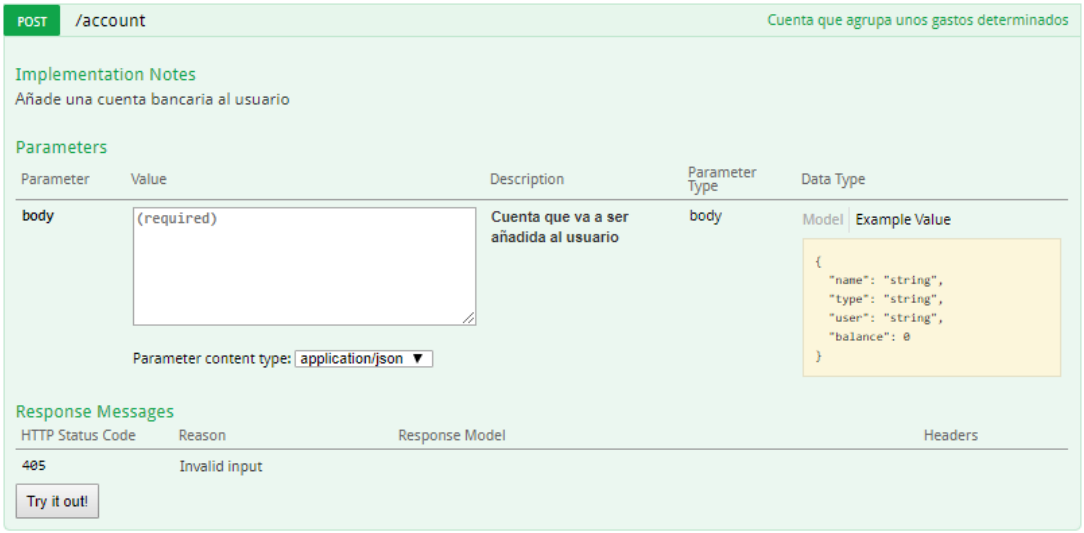

Ilustración 35 - Ejemplo de prueba del método POST de la entidad Cuenta

6.4 Front

El front ha sido la parte que más trabajo ha requerido.

6.4.1 DESARROLLO

Antes de explicar cómo hemos estructurado la parte el cliente, veamos un pequeño esquema en la siguiente imagen.

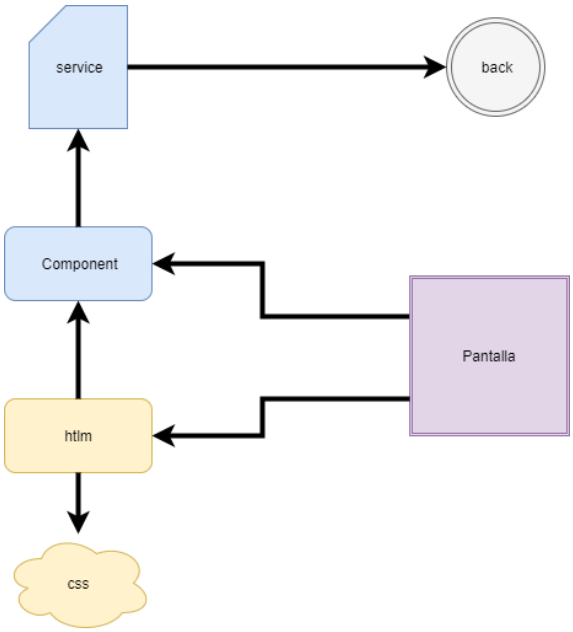

Ilustración 36 - Esquema estructura frontend

Nuestra estructura del front, comenzará en los fundamentos de **Angular 2**, puesto que cada **pantalla** de la aplicación está compuesta por dos archivos, el HTML que contendrá el código puramente visual y el archivo **component** que contendrá código JavaScript y hará de *controlador* de la vista.

Estructuración

Vamos a describir por partes cómo hemos implementado el frontend centrándonos sólo, con el fin de evitar extensiones innecesarias, en las clases que hemos considerado más importantes.

La imagen que a continuación vemos es la estructura principal de la parte de código perteneciente al frontend del proyecto.

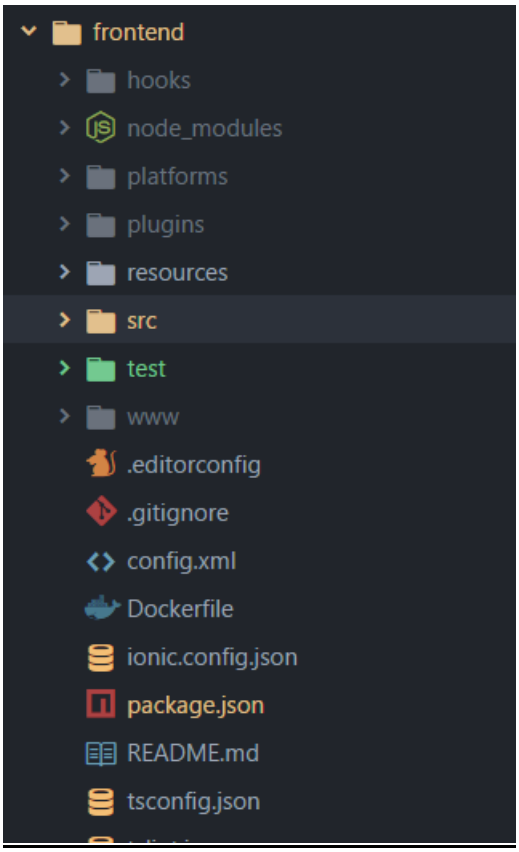

Ilustración 37 - Estructuración principal frontend

Nos centraremos de momento en la carpeta **src**, en ella se almacenarán las carpetas principales de nuestro código.

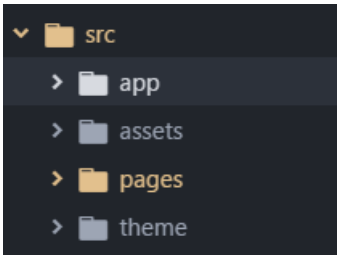

Ilustración 38 - Estructura carpeta src frontend

/app: Esta carpeta almacenará: las carpetas con las clases de las estructuras principales, los servicios que conectarán con el front y los validadores personalizados.

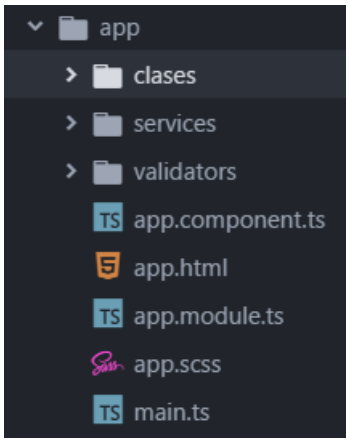

Ilustración 39 - Estructura carpeta app del frontend

/clases: Contiene las estructuras principales (cuenta, movimiento y usuario) que utilizaremos en los controladores. La clase *index* guarda los *imports* de todas las clases creadas para facilitar su implementación en los ellos.

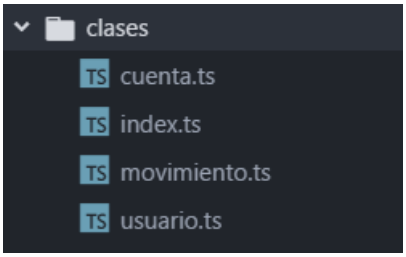

Ilustración 40 - Estructura carpeta clases del frontend

```
export class Cuenta {
  name:string;
  type:string;
  user:string;
  balance:number;

  constructor() {
    this.name = "";
    this.type = "";
    this.user = "";
    this.balance = 0;
  }
}
```

Ilustración 41 - Ejemplo clase Cuenta incluida en la carpeta clases

/services: Esta carpeta contendrá los servicios, estos son las clases que unirán nuestro front con nuestro back. Un controlador podrá hacer una llamada a una función perteneciente a un servicio concreto que a través de una petición URL se comunicará con el back y trabajará con los datos requeridos de nuestra **BBDD**.

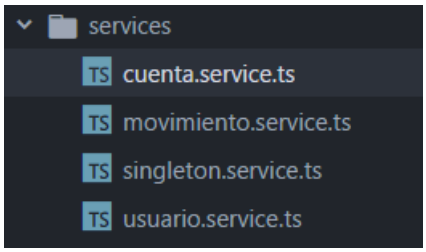

Ilustración 42 - Estructura carpeta services del frontend

Veamos un ejemplo a una llamada al back desde el front, en este caso para crear una cuenta:

```
urlCrearCuenta:string = "http://localhost:8080/account";
```

Ilustración 43 - Ejemplo variable de URL para una llamada al backend

```
createCuenta(cuenta:any) {
  let headers = new Headers({
      'Content-Type': 'application/json',
      'Access-Control-Allow-Origin': '*'
  });
  this.http.post(this.urlCrearCuenta, cuenta, { headers: headers }).subscribe( res =>{ res.json()} )
}
```

Ilustración 44 - Función que llama al backend de nuestra aplicación

App.modules.ts: Aquí declaramos todos los componentes, servicios, *imports* externos, proveedores...

```
//Servicios
import { CuentaService } from '../app/services/cuenta.service';
import { MovimientoService } from '../app/services/movimiento.service';
import { UsuarioService } from '../app/services/usuario.service';
import { SingletonService } from '../app/services/singleton.service';
```

Ilustración 45 - Ejemplo de importación de componentes creados en app.modules.ts

```
providers: [
  HttpModule,
  StatusBar,
  SplashScreen,
  CuentaService,
  UsuarioService,
  SingletonService,
  MovimientoService,
  EmailComposer,
  {provide: ErrorHandler, useClass: IonicErrorHandler}
]
```

Ilustración 46 - Ejemplo de declaración de los proveedores utilizados en el frontend en la clase app.modules.ts

App.scss: En este archivo hemos guardo todo el css utilizado en la aplicación.

```
//FONDOS
.background-general {
  background: #E0F2F7;
}

.background-navbar {
  color: #0080FF;
}

.fondo-blanco{
  background: #FFFFFF;
}
.fondo-verde {
    background:#ABEBC6
}
.fondo-verde-claro {
    background:#D5F5E3
}
.fondo-naranja-claro {
  background:#FAB899;
}
```

Ilustración 47 - Fragmento de la clase que contiene el css de la aplicación

/assets: En esta carpeta guardamos las imágenes del proyecto como ejemplo el icono.

/pages: En esta carpeta guardamos todo el código de las pantallas de la vista de *tapCuentas* ordenado por carpetas.

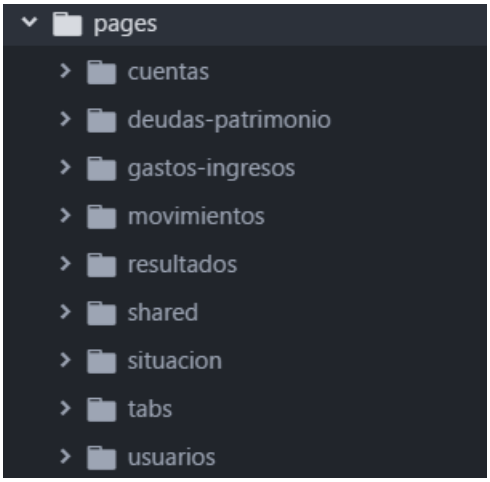

Ilustración 48 - Carpeta con el código de la vista

Algunas carpetas contienen otras subcarpetas, por ejemplo, en la carpeta *cuentas* están todas las subcarpetas pertenecientes a las pantallas que están relacionadas con las cuentas, la carpeta *movimientos* con todas las que tienen que ver con movimientos… etc

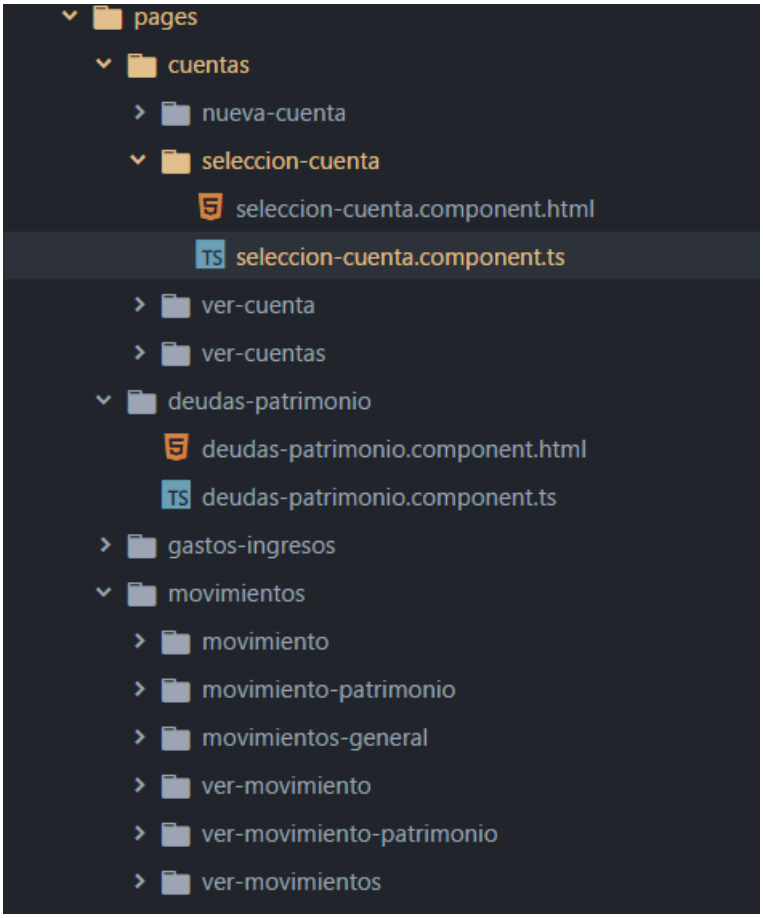

Ilustración 49 - Subcarpetas de la parte de la vista.

Veamos una carpeta almacenada en *pages* para explicarla como ejemplo.

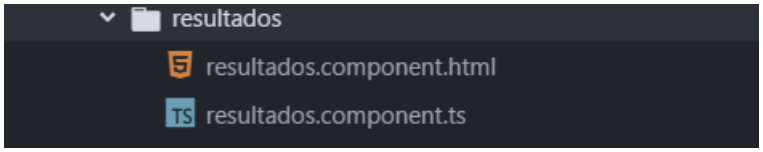

Ilustración 50 - Carpeta donde se encuentra el código de la pantalla resultados

En esta carpeta en concreto, se almacena el código de una de las pantallas de la aplicación dividido en dos archivos que la componen, como hemos mencionado en el primer punto, el archivo de la vista y el archivo del controlador.

resultados.component.html: aquí se almacena el código *HTML* y el código *Angular 2* con los componentes *Ionic* que formarán la vista.

resultados.component.html

```
                    <label class="texto-gris">{{ cuenta.saldo }}{{moneda}}</label>
                </ion-col>
                </ion-row>
            </ion-item>
          </div>
          <!--Si es una lista larga -->
          <div *ngIf="desplegableIngreso">
            <ion-item
            class="fondo-verde margen-cuentas-resultados"
              *ngFor="let cuenta of cuentasIngreso">
              <ion-row>
                <ion-col  col-9>
                    {{ cuenta.nombreCuenta }}
                </ion-col>
                <ion-col col-3 class="texto-derecha">
                    <label class="texto-gris" *ngIf="cuenta.saldo > 0">+</label>
                    <label class="texto-gris">{{ cuenta.saldo }}{{moneda}}</label>
                </ion-col>
                </ion-row>
            </ion-item>
          </div>
          <ion-item class = "text-center"*ngIf="cuentasIngreso.length > 2">
            <a (click) = "clickIngreso()">
             <ion-icon *ngIf="!desplegableIngreso" color = "primary" ios="ios-arrow-dropdown" md="md-arr
              <ion-icon *ngIf="desplegableIngreso" color = "primary" ios="ios-arrow-dropup" md="md-arrow
           </a>
          </ion-item>
      </ion-card-content>
  </ion-card>
```

Ilustración 51 -Captura resultados.component.html

resultados.component.ts: aquí se almacena el código TypeScript del controlador. Cargamos las librerías, trabajamos con los datos que nos vienen de los controladores de los servicios o del controlador de la vista que lo ha llamado y las funciones de los elementos de la vista.

resultados.component.ts

```
import { Component, OnInit } from '@angular/core';
import { CuentaService } from '../../app/services/cuenta.service';
import { SingletonService } from '../../app/services/singleton.service';
import { NavController, NavParams } from 'ionic-angular';

@Component({
  selector: 'app-resultados',
  templateUrl: 'resultados.component.html',
})
export class ResultadosComponent implements OnInit {
  cuentasIngreso:any[] = [];
  cuentasGasto:any[] = [];
  existenIngreso:boolean = false;
  existenGasto:boolean = false;
  totalIngresos = 0;
  totalGastos = 0;
  resultado = 0;
  moneda = "€"
  situacion:string ="";
  desplegableIngreso:boolean = false;
  desplegableGasto:boolean = false;
  usuario = "";

  constructor(public serviceSingleton:SingletonService,
              public navParams:NavParams,
              public _serviceCuenta:CuentaService,
              public navCtrl:NavController) {  }

  calculaTotal() {
    this.resultado = this.totalIngresos - this.totalGastos;
    if(this.resultado > 0) {
        this.situacion ="AHORRO";
    }else if(this.resultado == 0){
      this.situacion ="RESULTADO";
    } else {
      this.situacion ="DÉFICIT";
    }
```

Ilustración 52 - Captura del controlador de la pantalla Resultados

Los controladores también son los encargados de llamar de unas vistas a otras.

6.4.2 EJECUCIÓN

Para ejecutar la parte del front únicamente debemos ejecutar el servidor Ionic:

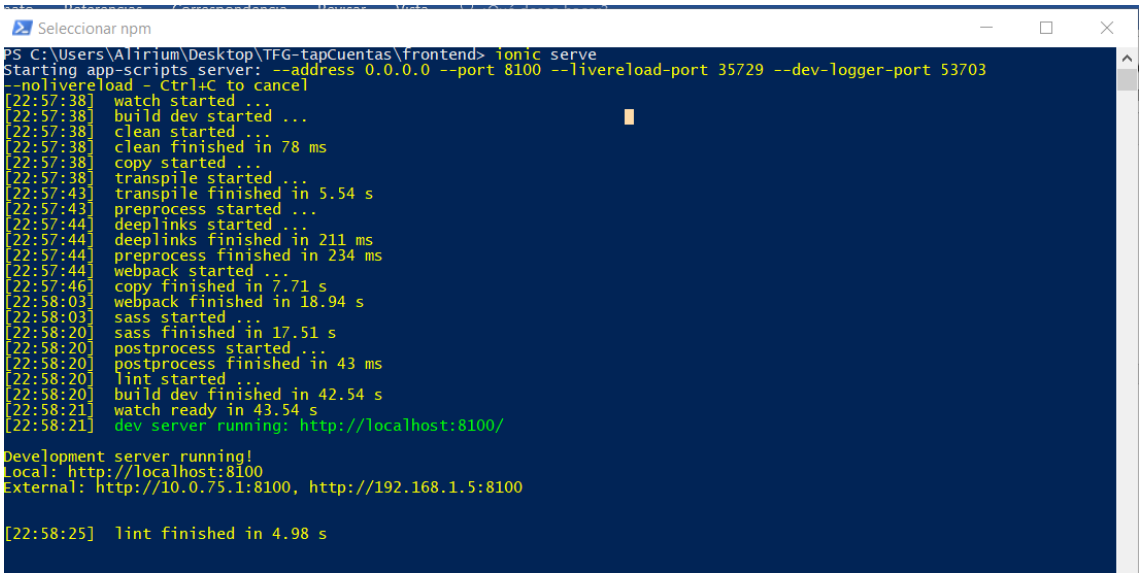

Ilustración 53 - Ejecución de la parte front con Ionic

Ya podríamos acceder desde el navegador con la URL de localhost:

http://localhost:8100/

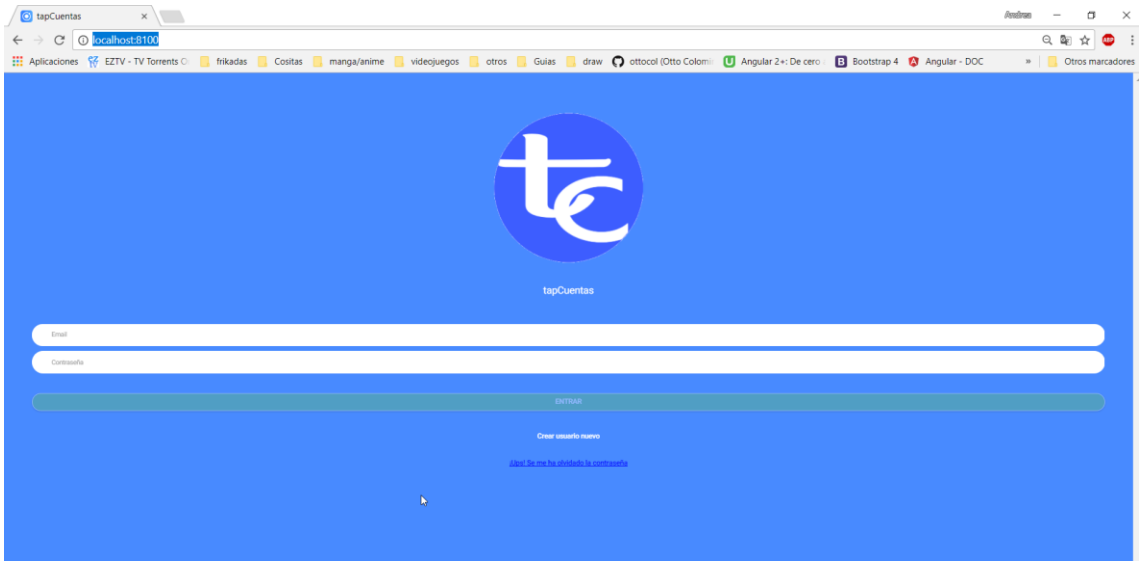

Ilustración 54 - http://localhost:8100/

6.4.3 TESTING

Para hacer testing sobre la parte del front, hemos hecho, por una parte, tablas de casos de prueba donde se describe una situación y cómo debe responder la aplicación.

CASO DE PRUEBA	ESCENARIO	RESULADO ESPERADO	RESULTADO REAL
C5	El usuario accede a realizar un Ingreso, señlecciona la cuenta de 'Trabajo' y en el formulario rellena con -5 en campo importe	La aplicación muetsra el mensaje de Error ' Introduce una cantidad mayor a 0'	OK
C6	El usuario accede a realizar un Ingreso, señlecciona la cuenta de 'Trabajo' y en el formulario deja vacío el campo importe	La aplicación muetsra el mensaje de Error 'La cantidad no puede estar vacía'	OK

Ilustración 55 – Fragmento tabla de casos de prueba

Y por otra parte hemos hecho uso de pequeños test unitarios utilizando **Jasmine** y **karma.** Jasmine es lo que hemos utilizado para crear los test, y Karma ha sido el encargado de ejecutarlos. Estos test se han utilizado sobre todo para probar os services.

Todos estos test han seguido la misma estructura.

```
1  import { async, TestBed } from '@angular/core/testing';
2  import {CuentaService} from  '../src/app/services/cuenta.service'
3
4
5  var servivioCuentas:CuentaService;
6
7  describe('Cuentas servicio pruebas', () => {
8    beforeEach(() => {
9         var usuario:string = 'andrea@ua.es';
10      });
11
12     it('Coge las cuentas por usuario', () => {
13           let result = servivioCuentas.getCuentas(usuario);
14           expect(result.length).toEqual(3);
15        }
16   ));
```

Ilustración 56 - Ejemplo test con Jasmine

Para ejecutar los test se ha utilizado el comando `npm test`

6.5 Docker

Hemos implementado Docker como forma de distribución de la aplicación, se trata de 'contenedores' donde podemos encapsular y preparar un determinado código para facilitar el despliegue de la app cuando se quiera poner en servicio, o también poder seguir desarrollando sin tener que instalar las dependencias.

¿Cómo funciona? Cuando ejecutamos un Docker, esta monta internamente una máquina virtual donde ejecutará los comandos que se le han indicado en la configuración. Una vez ejecutado, se tendrá acceso al contenido desde su máquina principal.

Los contenedores creados han sido subidos a la plataforma oficial de Docker, el enlace a estos es el siguiente: https://hub.docker.com/u/alirium/

Cuando se tiene la parte del código se quiere 'encapsular' lo primero es construir el contenedor, para ello creamos el archivo **Dockerfile**, este archivo es un documento de texto que contiene todos los comandos para configurar el *Docker*, básicamente es el encargado de **crear el contenedor**.

Pata ejecutar dicho archivo ejecutamos el comando:

```
docker build -t alirium/tapcuentas-back:1.0 .
```

Una vez construido, podemos subirlo opcionalmente a la plataforma con el comando:

```
docker push alirium/tapcuentas-front:1.0
```

Para bajarse el contenedor, el comando sería:

```
docker pull alirium/tapcuentas-front:1.0
```

Una vez tenemos el contenedor, lo siguiente es ejecutarlo, para ello tenemos el archivo **docker-compose.yml** donde se especifican los comandos necesarios para que funcione.

Para ejecutar este archivo haremos uso del comando:

```
docker-compose up -d
```

Este comando debe ser ejecutado en la careta que contenga el archivo docker-compose. El '-d' indica que lo haremos en un segundo plano.

En resumen, una vez tengamos el contenedor y ejecutemos este último comando, el *docker* estará listo, es decir, ya tendremos el código ejecutándose.

6.5.1 BACK

Veamos cómo hemos configurado el back:

Primero creamos el contenedor con el *Dockerfile*, en él configuramos el sistema operativo de la máquina virtual, en este caso *alpine*, y los comandos que instalarán las dependencias del *back*.

Dockerfile

```
# FROM debian:8.6
FROM node:6-alpine
MAINTAINER Andrea Ruiz Macia <arm102@alu.ua.es>
# Actualizamos el repositorio de debian
# RUN apt-get install -y curl

RUN apk add --update bash && rm -rf /var/cache/apk/*

RUN npm install -g swagger

ENV TZ=Europe/Madrid
RUN ln -snf /usr/share/zoneinfo/$TZ /etc/localtime && echo $TZ > /etc/timezone
RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app
# copiamos el proyecto en el directorio /usr/src/app e instalamos librerias con npm
COPY . /usr/src/app
WORKDIR /usr/src/app
RUN npm install
# exponemos los puertos al exterior para que sean accesibles desde nuestro host
# DEMO

EXPOSE 8080
EXPOSE 27017

WORKDIR /usr/src/app

# Eliminamos retornos de carro del fichero creado en windows
RUN sed -i -e 's/\r$//' init.sh
#CMD ["/bin/bash"]

```

Ilustración 57 - Archivo dock Dockerfile del back

En este Dockerfile, además, añadimos el puerto que usará la base de datos.

Segundo, creamos el archivo *docker-compose* para ejecutar el contenedor creado. Contiene además la dependencia con la base de datos para que esta se ejecute junto al *back* una vez lancemos el contenedor.

```
version: '2'
services:
 swagger: #Servicio Backend tfg-api
  image: alirium/tapcuentas-back:1.0
  #restart: always
  #tty: true
  command: npm start
  ports:
   - '8080:9080'
  links:
   - db
 db:
  image: mongo:3.2
  #restart: always
  volumes:
   - C:\Users\Alirium\Documents\dockerData:/data/db/
  ports:
   - '27017:27017'
  command: mongodb
```

Ilustración 58 – Configuración del archivo docker-compose del backend

Una vez ejecutamos el *docker*, podemos acceder al *back* de nuestra aplicación.

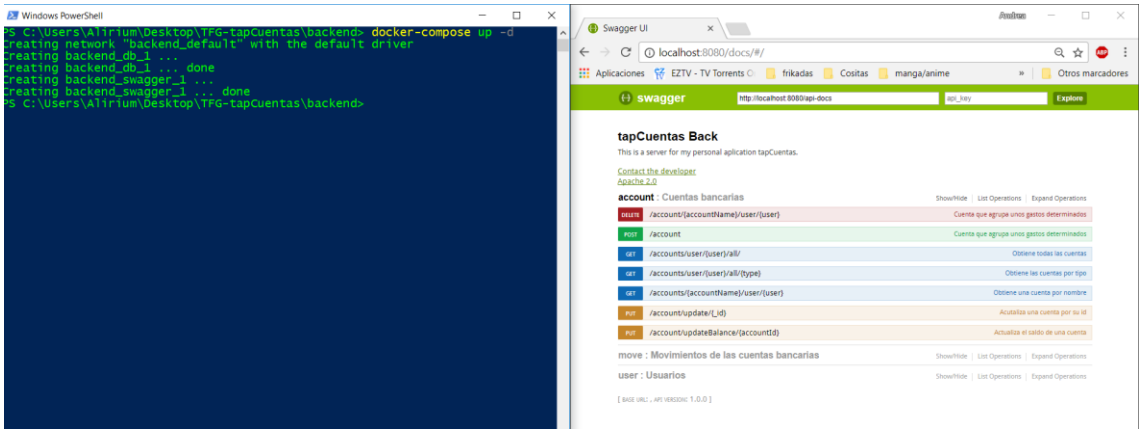

Ilustración 59 - - Ejecución docker backend

6.5.2 FRONT

Ahora veamos cómo hemos implementado *docker* el front:

Primero creamos el contenedor con el *Dockerfile*, en él configuramos el sistema operativo de la máquina virtual, en este caso *alpine*, y los comandos que instalarán las dependencias del *front*.

Dockerfile

```
# FROM debian:8.6
FROM node:8-alpine
MAINTAINER Andrea Ruiz Macia <arm102@alu.ua.es>
# Actualizamos el repositorio de debian
# RUN apt-get install -y curl

RUN apk add --update bash && rm -rf /var/cache/apk/*

RUN npm install @angular/cli
RUN npm install -g ionic cordova

ENV TZ=Europe/Madrid
RUN ln -snf /usr/share/zoneinfo/$TZ /etc/localtime && echo $TZ > /etc/timezone
RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app
# copiamos el proyecto en el directorio /usr/src/app e instalamos librerias con npm
COPY . /usr/src/app
WORKDIR /usr/src/app
RUN npm install
# exponemos los puertos al exterior para que sean accesibles desde nuestro host
# DEMO

EXPOSE 8100

WORKDIR /usr/src/app

# Eliminamos retornos de carro del fichero creado en windows
RUN sed -i -e 's/\r$//' init.sh
#CMD ["/bin/bash", "init.sh"]
```

Ilustración 60 - Configuración archivo Docker del front end

Segundo, creamos el archivo *docker-compose* para ejecutar el contenedor creado. Contiene el comando que ejecutará el código del front:

Ilustración 61 - Configuración archivo docker-compose frontend

Una vez ejecutamos el *docker*, podemos acceder al *front* de nuestra aplicación.

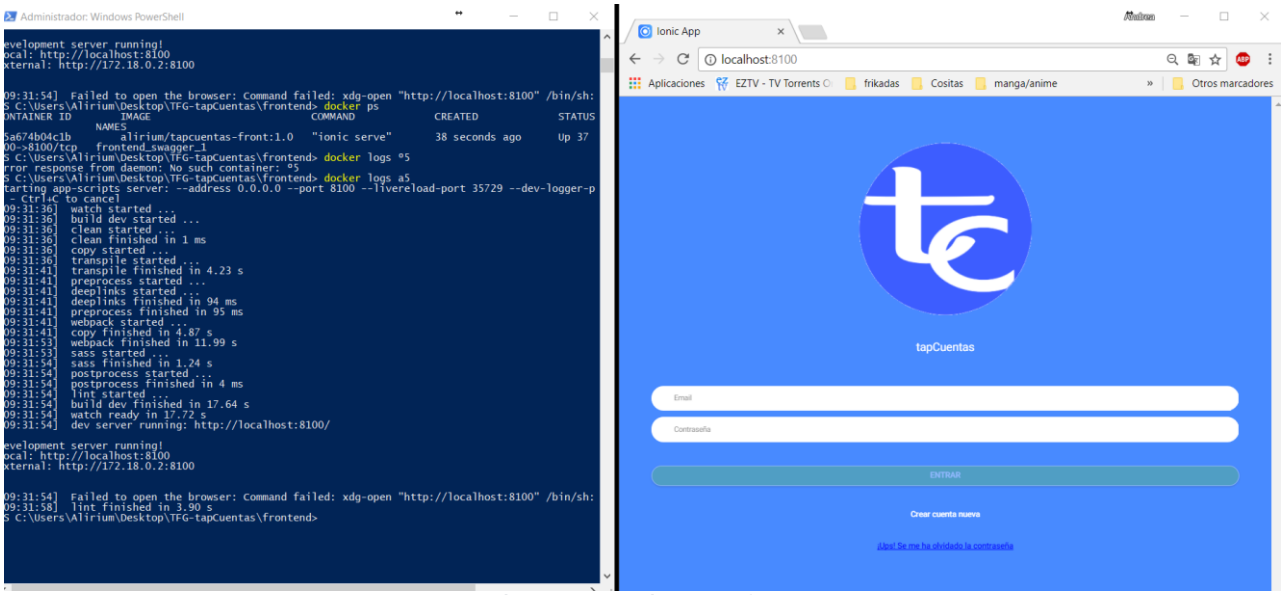

Ilustración 62 - Ejecución docker frontend

7. Funcionalidades de la aplicación

Un usuario va a poder hacer las siguientes funcionalidades:

- Registrarse
- Hacer Login
- Cerrar sesión
- Recuperar contraseña
- Modificar usuario
- Borrar Usuario
- Crear cuenta Ingreso
- Crear movimiento Ingreso
- Crear cuenta Gasto
- Crear movimiento Gasto
- Crear cuenta Patrimonio
- Crear movimiento Patrimonio
- Crear cuenta Deuda
- Crear movimiento Deuda
- Ver resultado financiero actual
- Ver situación financiera actual

8. Futuras mejoras

tapCuentas ha sido desarrollada en su primera versión y claro está que de cara al futuro tiene aspectos a mejorar y aspectos a crecer para una mejor experiencia del usuario. Trabajar y desarrollar una aplicación no es fácil por lo que en esta primera versión nos ceñimos a aprender el funcionamiento de las tecnologías usadas y funcionalidades básicas. De cara a una segunda versión, creo que sería recomendable e interesante incluir las siguientes mejoras:

Añadir más seguridad

Este es uno de los aspectos se hubiese gustado incluir, pero no entraba dentro de los sprints planificados por el tiempo ajustado. Se pretende incluir el estándar **JSON Web Token** para una mayor seguridad entre peticiones.

Contabilidad de doble asiento

También sería un reto hacer tapCuentas un poco más compleja dotándola del enfoque tradicional del doble asiento. Cada movimiento realizado tendría que tener su contrapartida de compensación en otra u otras cuenta/s distinta /s, formando Asientos que cuadren entre sí.

Más filtros de búsqueda

Sería muy interesante que la aplicación pudiera presentar la evolución en el tiempo (Años o Meses), de las Cuentas, los Resultados y la Situación

Distribución en los markets

Una vez tuviéramos la aplicación con todos los aspectos deseados incluidos, se estudiará el 'subirla' a las plataformas de tiendas móviles como la 'Store app' de Android o la 'Apple store' de IOS.

Offline

Esta primera versión está hecha para funcionar únicamente con una conexión a internet activa, Sería interesante estudiar un comportamiento *offline*.

9. Conclusión

Personalmente, emprender este proyecto desde cero y crear esta primera versión de tapCuentas ha supuesto todo un reto para mi persona. Ha habido momentos difíciles, pero al final con investigación y paciencia he conseguido llevar la aplicación adelante.

De este TFG he aprendido nuevas tecnologías como me propuse como Angular 2, Ionic, Docker, Swagger, MongoDB entre otras que, aunque me llevo **una curva de aprendizaje** larga al principio, poco a poco he ido desenvolviéndome con mucha ms facilidad ante los retos que me he ido encontrando.
También he mejorado mucho en la organización de proyectos a nivel personal que podré enfocar más adelante en el mundo laboral.

Estoy orgullosa de poder presentar este proyecto que, aunque está aún en su primera versión, tiene mucho que crecer.

Gracias por leer esta memoria que está llena de tiempo de ilusión y trabajo.

10. Anexo 1: Pantallas tapCuentas

A continuación, veremos cómo ha quedado la primera versión de tapCuentas después de su desarrollo.

10.1 Login

10.1.1 PANTALLA LOGIN

Cuando el usuario abra la pantalla, la primera pantalla que va a ver es la pantalla de **login,** en esta pantalla se podrá *loguear* con su email y contraseña.

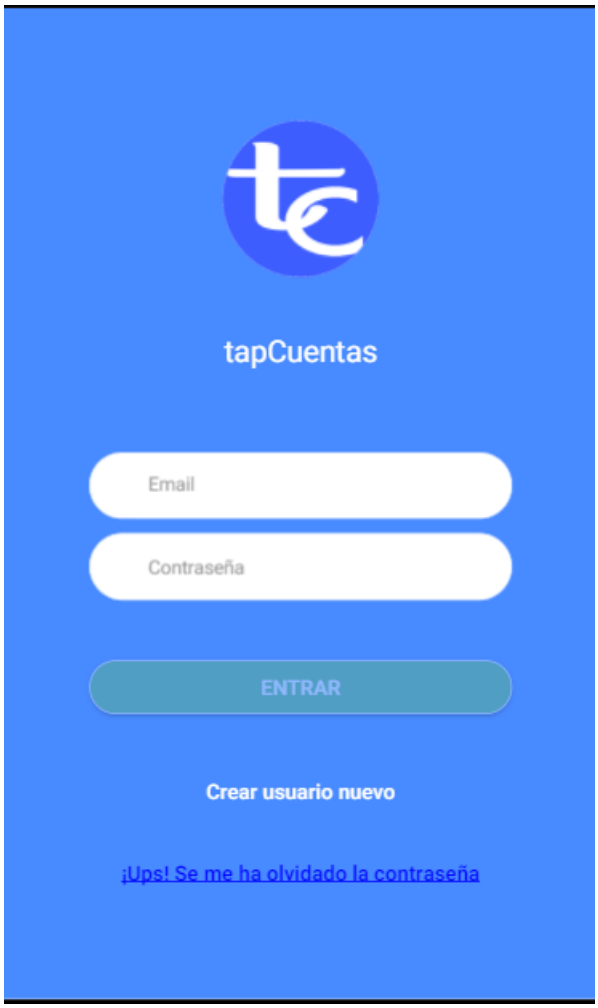

Ilustración 63 - Pantalla principal login

Todos los formularios pertenecientes a las pantallas de este primer apartado **5.1**, tendrán un formato concreto que no permitirá al usuario dar de alta al formulario si no se han rellenando los datos, bloqueando el botón de darlo de alta.

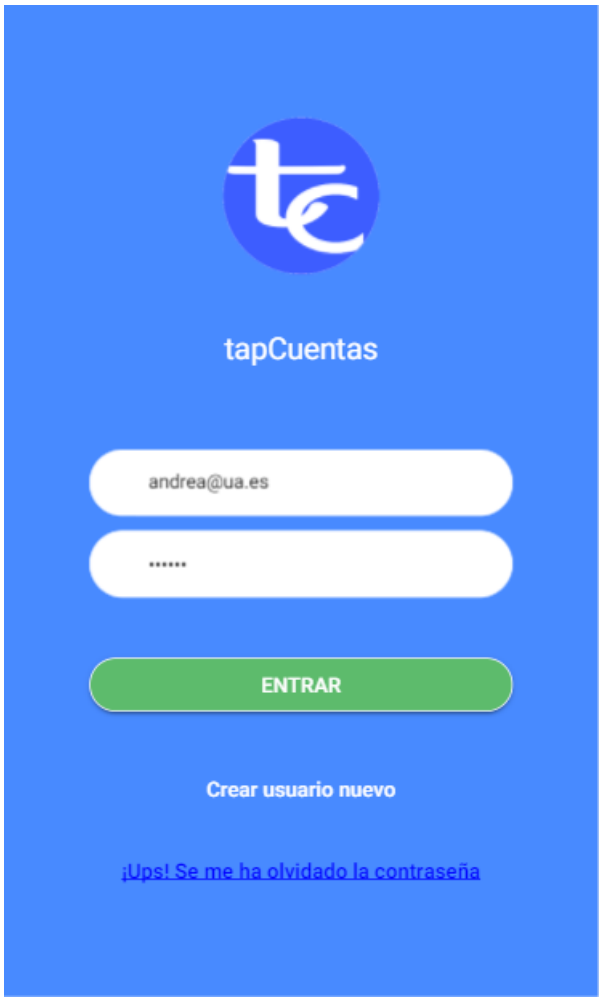

Ilustración 64 - Pantalla login con credenciales

Si el usuario introducido no existe, la aplicación mostrará un mensaje de error avisando al usuario.

Ilustración 65 - Error de usuario en la pantalla login

Si el usuario es correcto pero la contraseña no, se mostrará el mensaje correspondiente.

Ilustración 66 - Error de contraseña en pantalla login

10.1.2 PANTALLA REGISTRO

Si el usuario es nuevo, desde la pantalla *login* puede acceder al enlace de registro, en esta pantalla el usuario se da de alta en el sistema introduciendo su email y contraseña, existe una verificación donde debe escribir dos veces estos campos para asegurar que escribe correctamente las futuras credenciales que desean.

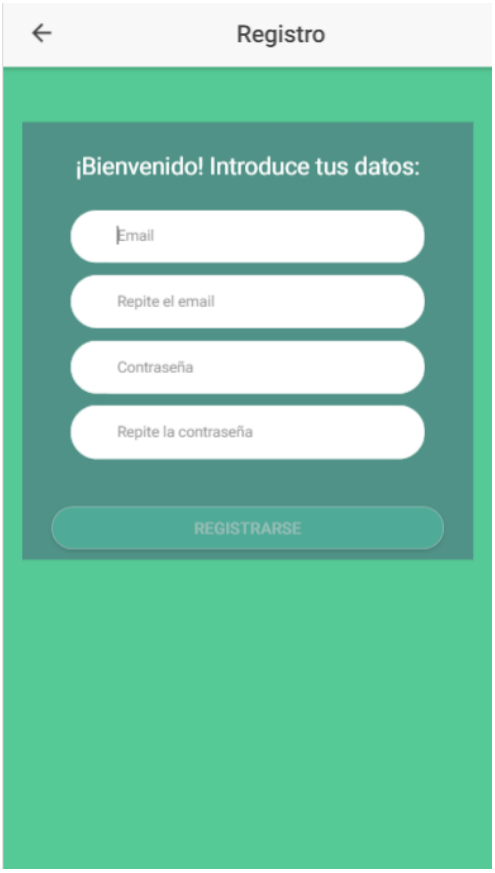

Ilustración 67 - Pantalla registro usuario

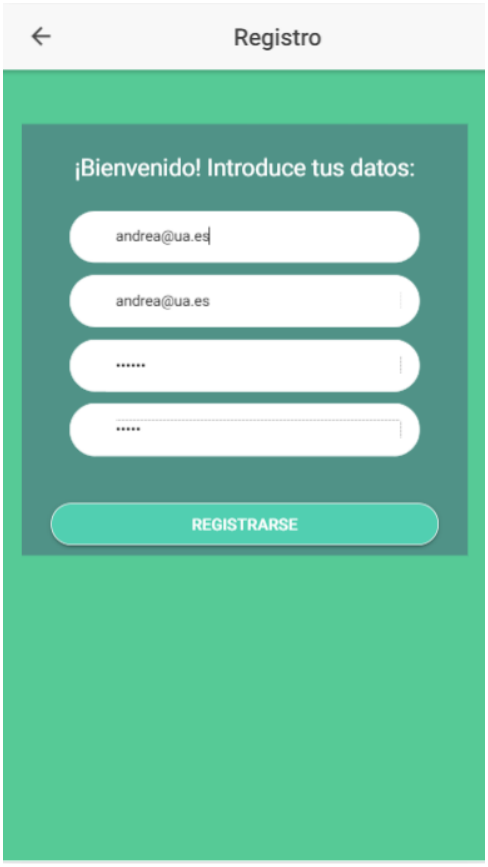

Ilustración 68 - Pantalla registro usuario con credenciales

Al igual que en la pantalla de login, en esta pantalla también existe verificación. En el caso del campo del email comprueba si el formato es válido, además tiene dos campos de verificación de email y contraseña que comprueba si coinciden entre ellos para evitar errores del usuario a la hora de escribirlos.

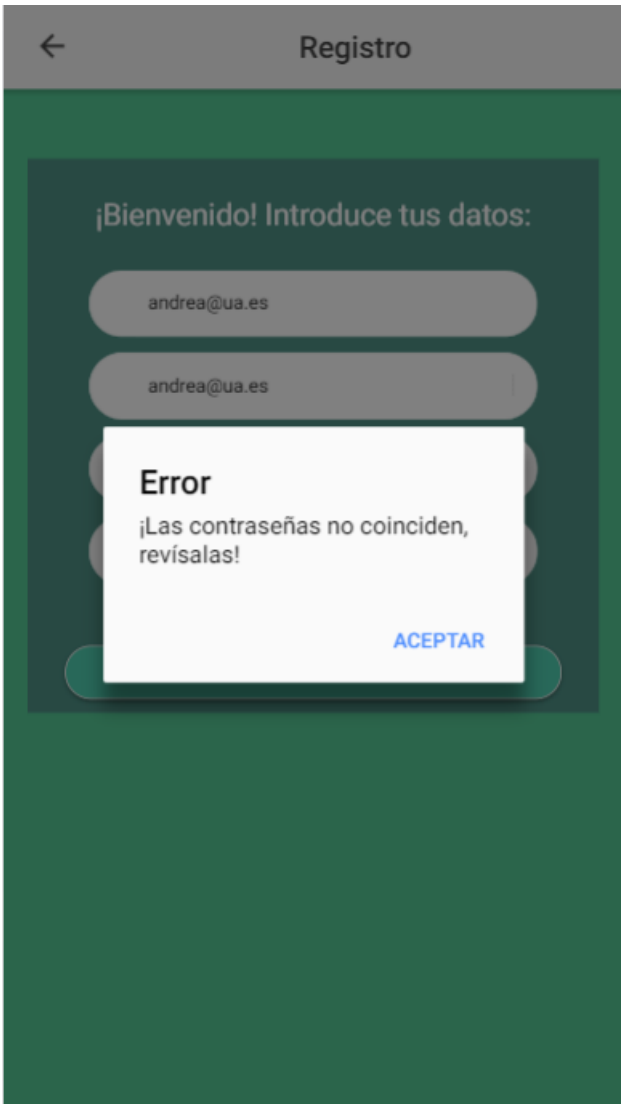

Ilustración 69 - Error comprobación datos pantalla registro

También se comprueba que el email no esté previamente ya registrado.

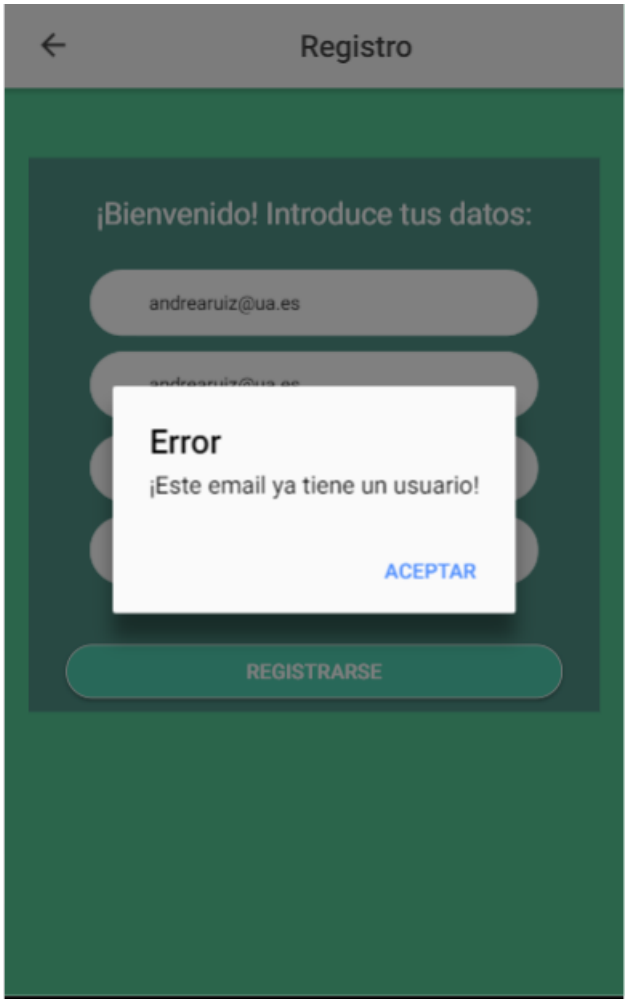

Ilustración 70 - Error registro email ya registrado

Si el registro se completa correctamente, la aplicación se redirige a la pantalla de *login* y muestra al usuario un mensaje de aviso de que todo ha ido correctamente.

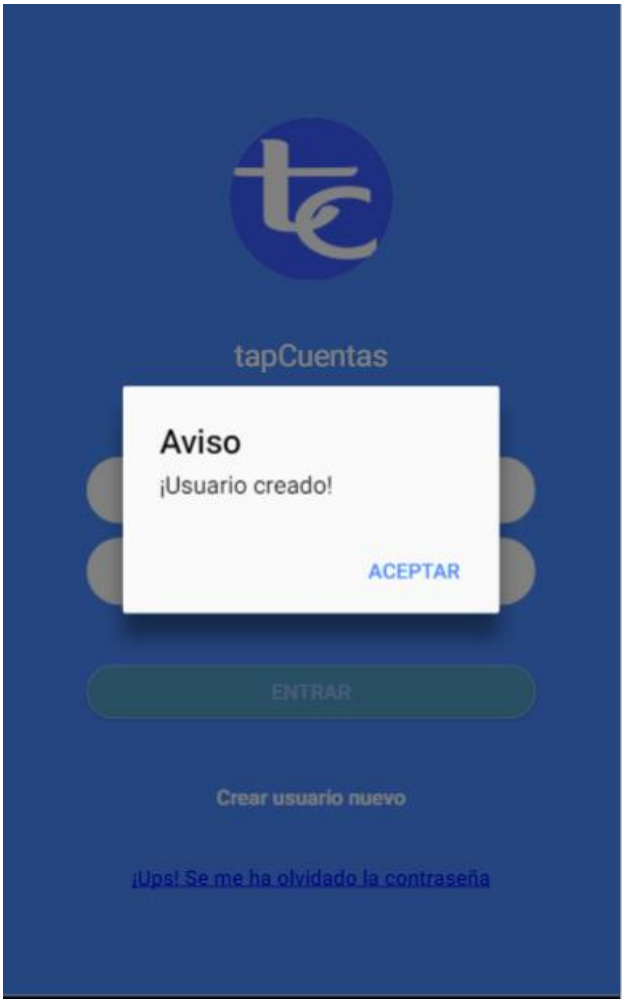

Ilustración 71 -Cuenta de usuario creada correctamente

10.1.3 PANTALLA RECUPERACIÓN

Si al usuario se le hubiese olvidado la contraseña, en la pantalla principal *login* hay un enlace en el que este puede pinchar y se le redirigirá a la pantalla de recuperación donde tapCuentas le pedirá su email de usuario al que mandará la contraseña asociada. Esto es posible gracias a uno de los componentes de los que Ionic 2 provee, en concreto **Email Composer**, con el que podemos enviar un email al usuario recordándole las credenciales personales.

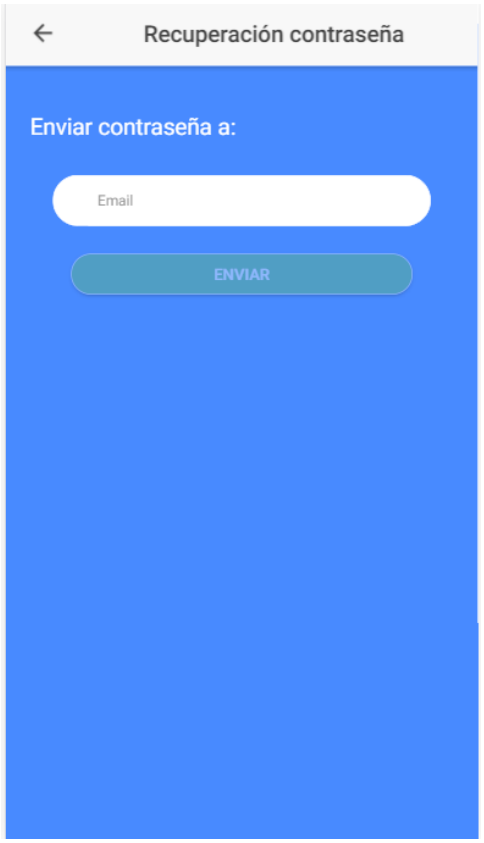

Ilustración 72 - Pantalla recuperación contraseña

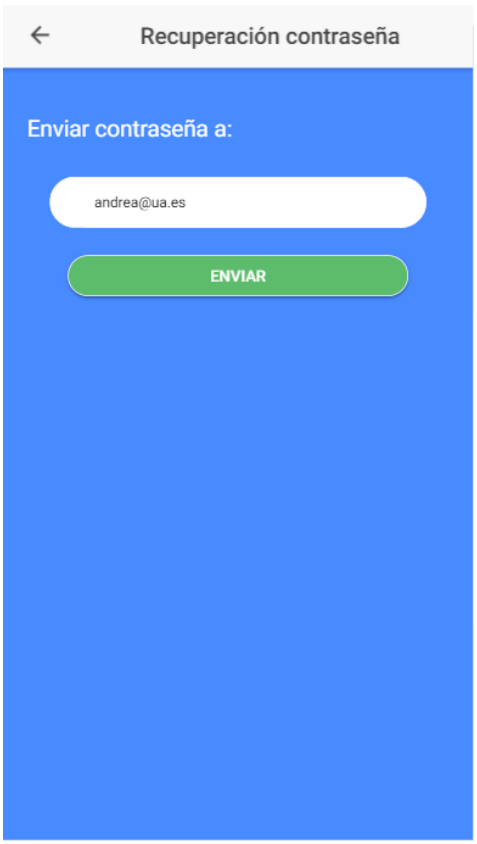

Ilustración 73 - Pantalla recuperación con datos

Como hemos dicho en las anteriores pantallas, esta también tendrá una verificación de si el email introducido existe o no en la base de datos.

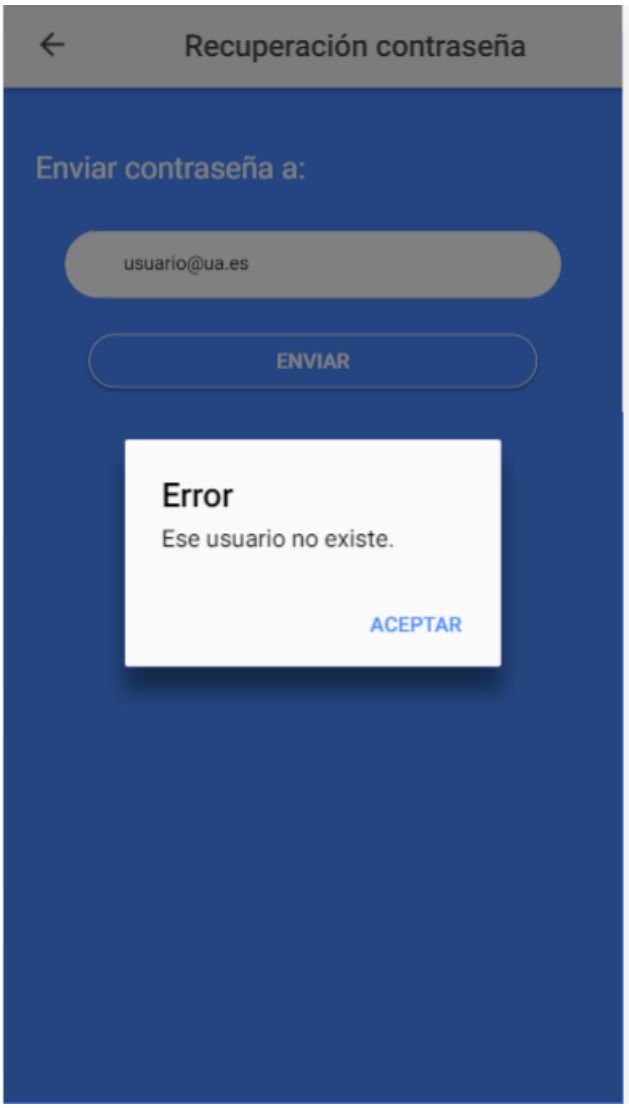

Ilustración 74 - Error usuario en pantalla recuperación credenciales

Si todo ha ido correctamente, la aplicación se redirige a la pantalla *login* y se le muestra un mensaje al usuario de que el email ya ha sido enviado.

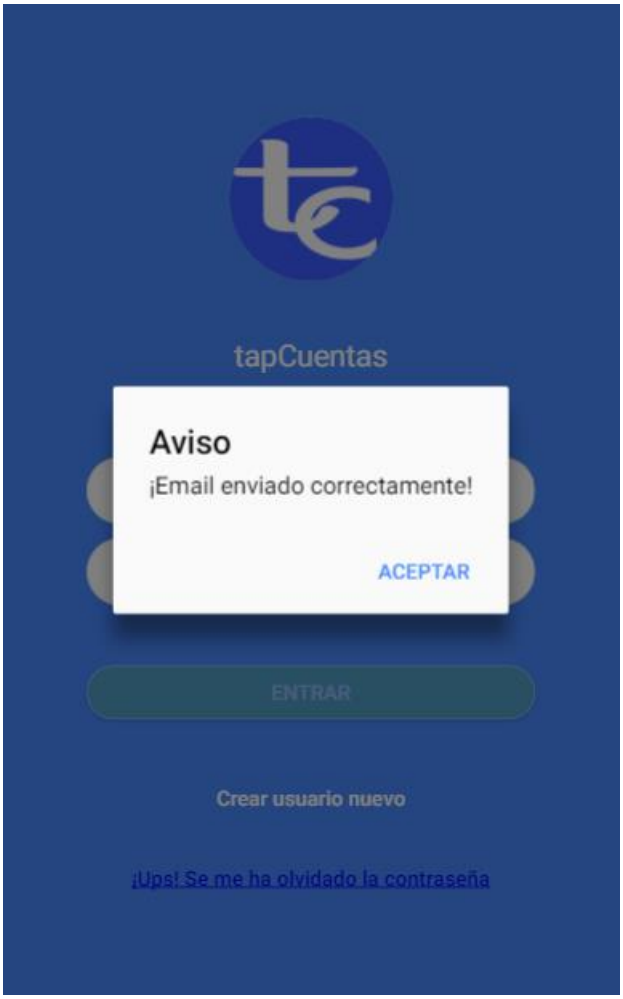

Ilustración 75 - Mensaje aviso de que todo ha ido correcto en la recuperación de contraseña

10.2 Sección Inicio

Cuando el usuario se loguea, su usuario queda almacenado en una variable global que nos mostrará toda la información únicamente asociada a ese usuario. Es la primera pantalla que veamos después de *loguearnos* que contendrá dos secciones que contaran con las siguientes opciones:

- Contabilizar de cualquiera de los dos tipos principales
- Ver las cuentas de los dos tipos principales
- Ver los movimientos de los dos tipos principales

Ilustración 76 - Sección inicio en la barra inferior de navegación

10.3 Contabilizar

Cuando un usuario se *loguea* correctamente accede a la pantalla de inicio que se divide en dos secciones, la primera, que es la el usuario ve, es la sección **Ingreso/Gasto** donde podremos contabilizar un Ingreso o un Gasto y ver las cuentas o movimientos asociados a estas transacciones.

Además, en todo momento veremos una barra de navegación situada en la parte inferior de la pantalla con tres iconos, el primero, siendo este una casa con el título **Inicio** siempre nos redirigirá a esta primera pantalla de contabilizar **Ingresos/Gastos**

Ilustración 77 - Pantalla inicio sección Ingreso/Gasto

Como podemos ver debajo del título de la aplicación de la pantalla, podemos acceder a la segunda sección simplemente tocando en el segundo bloque **PATRIMONIO/DEUDA**, lo mismo si quisiéramos volver al primero.

En esta segunda sección tenemos las mismas opciones que en la primera, pero con las opciones de Patrimonio o Deuda.

Ilustración 78 - Pantalla inicio sección Patrimonio/Deuda

10.3.1 INGRESO, GASTO, PATRIMONIO Y DEUDA

Cuando accedemos a crear uno de estos movimientos, la mecánica y la parte visual es similar, por lo que juntaremos su explicación en este punto.

Cuando accedemos a un movimiento, la siguiente pantalla a la que accedemos es la de seleccionar una cuenta en la que hacer el movimiento o en el caso de que no existiera o lo necesitáramos crear una nueva.

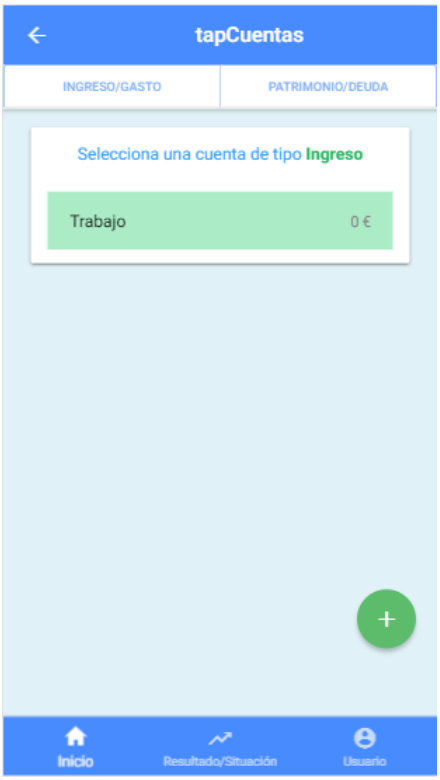

Ilustración 79 - Vista selección cuentas ingreso

Ilustración 80 - Vista selección cuentas gasto

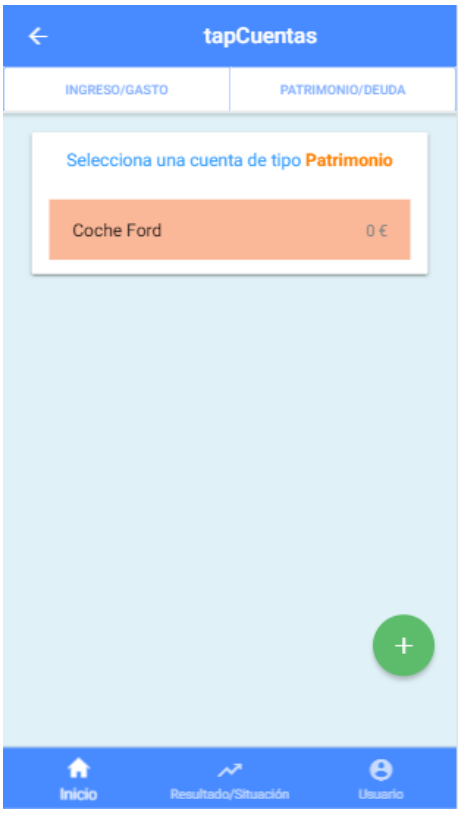

Ilustración 81 - Vista selección cuentas patrimonio

Si no existieran cuentas, la vista en cualquier tipo de movimiento seria la siguiente:

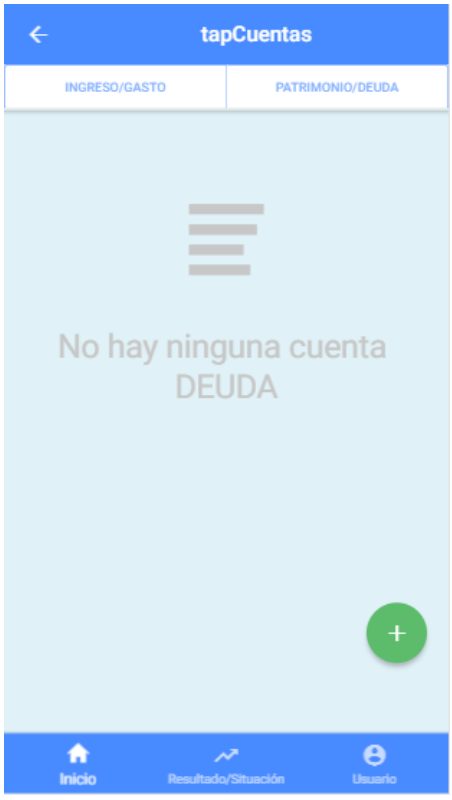

Ilustración 82 - Vista sin cuentas tipo deudas creadas

Para acceder a crear una nueva cuenta, presionamos al botón con el símbolo '+' que se encuentra situado en la zona inferior derecha de la pantalla.

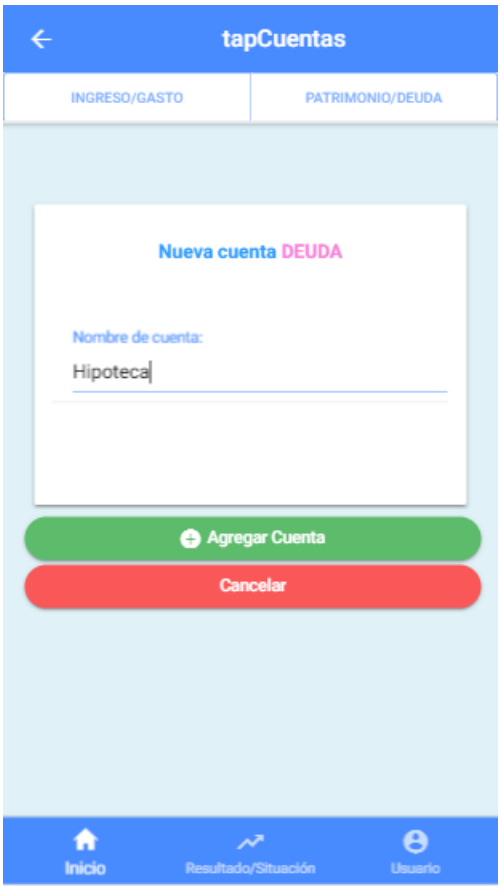

Ilustración 83 - Creación nueva cuenta tipo deuda

Cuando creamos una cuenta también se dispone de un control de validaciones, como que no esté en blanco, evitar espacios en blanco o que no se repita el nombre de la cuenta, puesto que para evitar confusiones esto no se permitirá, cada cuenta debe tener su propio nombre.

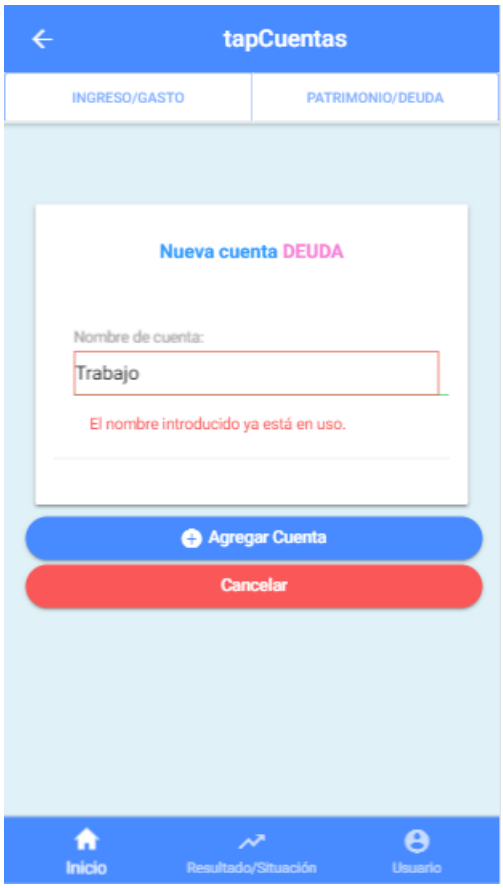

Ilustración 84 - Error nombre de cuenta existente vacío

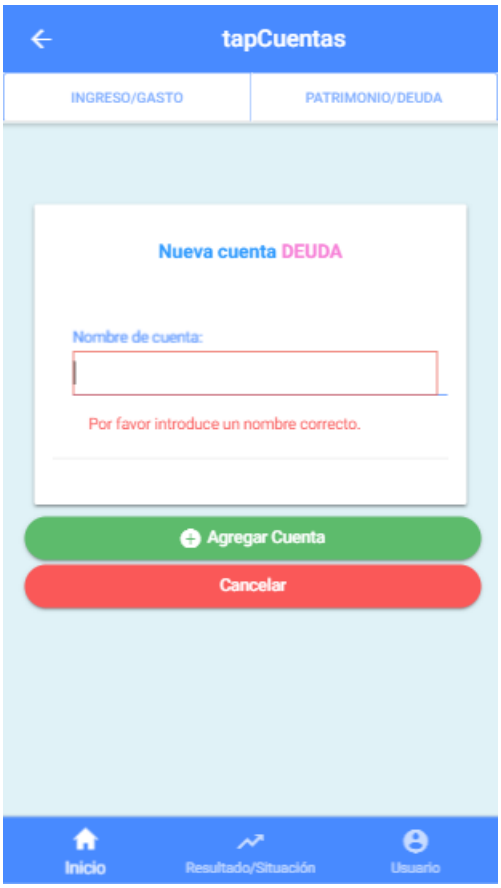

Ilustración 85 - Error campo nombre de cuenta

Si introducimos un nombre correcto, volveremos a la pantalla de selección de cuentas actualizada con la cuenta nuevamente creada acompañada de un mensaje de aviso al usuario de que todo ha ido correctamente, si por el contrario le damos al botón **cancelar**, volveremos a esta misma pantalla pero con las cuentas que hubiera ya creadas.

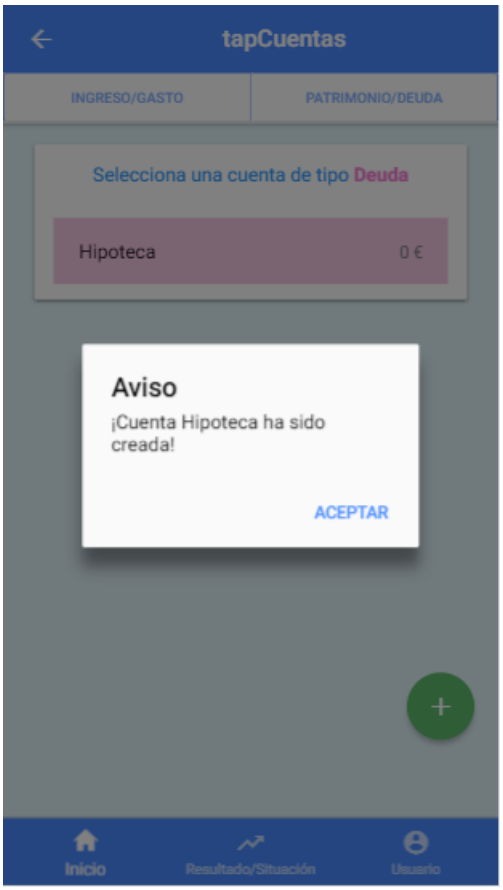

Ilustración 86 - Cuenta deuda creada correctamente

Cuando seleccionamos una cuenta presionando sobre su nombre, entramos en la pantalla de creación del movimiento deseado, esta será exactamente igual en los movimientos Ingreso y Gasto a diferencia del nombre del movimiento que se muestre en la cabecera del formulario.

El nuevo movimiento se compondrá de la siguiente entrada de datos:

- En nombre de la cuenta
- La fecha de creación del movimiento (Por defecto, la actual)
- El importe
- Un concepto que identifica el movimiento

Ilustración 87 - Pantalla nuevo movimiento tipo ingreso

Ilustración 88 - Pantalla nuevo movimiento tipo ingreso con datos

La fecha por defecto será la del día y hora actual, pero si pulsamos sobre ella, podremos cambiarla, con la ayuda de un calendario.

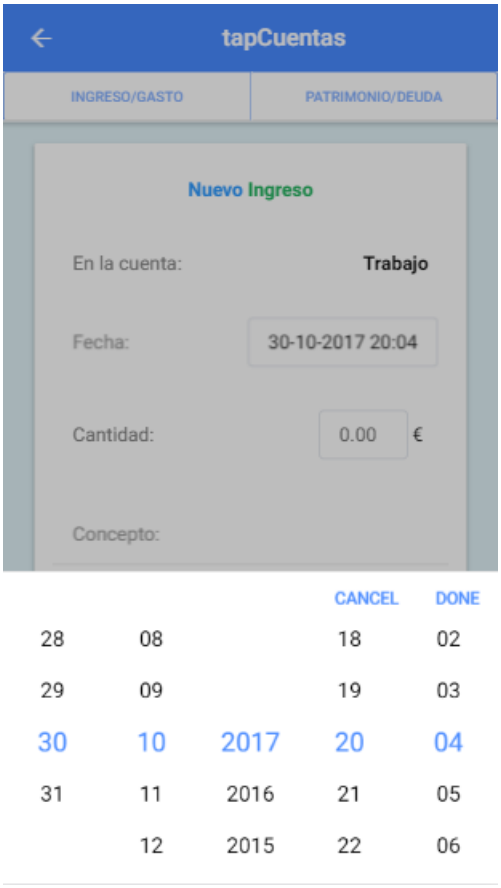

Ilustración 89 - Modificación campo fecha en movimiento

Esta pantalla también contará con validaciones como, por ejemplo, que la cantidad no esté vacía, que sea un número o que sea mayor que 0.

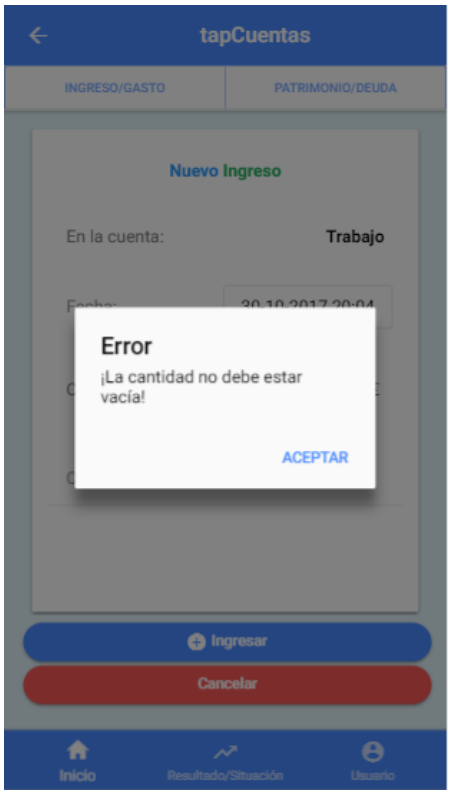

Ilustración 90 - Error de campo vacío obligatorio

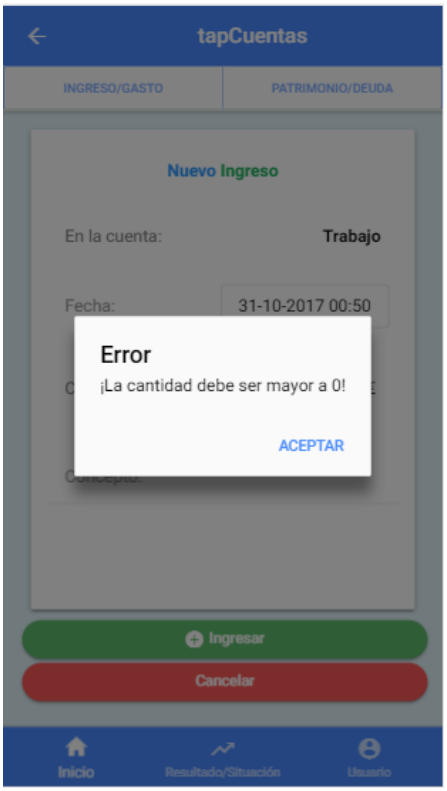

Ilustración 91 - Error de campo importe <= 0

Si se introdujeran correctamente los datos y se censa el movimiento, volveríamos a pantalla principal **Ingreso/Gasto** o **Patrimonio/Deuda** según correspondiera acompañado de un mensaje de aviso al usuario de que todo ha ido correctamente y con el saldo de la cuenta actualizado, por el contrario, si le diéramos al botón **cancelar**, volveríamos a la pantalla de *selección de cuentas*.

Ilustración 92 - Aviso movimiento Ingreso creado correctamente

En el caso de tratarse de un movimiento de tipo **patrimonio** o **deuda**, excepcionalmente el usuario podría elegir si se desea incrementar o **disminuir** el movimiento en la cuenta seleccionada. Para estos casos, la pantalla de nuevo movimiento contará con dos *radiobuttons* extra donde se elegirá si aumentar o disminuir la cantidad.

Esta funcionalidad, tiene ciertas validaciones.

- Si el saldo total de la cuenta es 0, no se podrá acceder a la opción de disminuir.
- Si existe saldo, nunca se podrá disminuir una cantidad menor al total del saldo actual.

Veamos un ejemplo, tenemos una cuenta recién creada de tipo **Patrimonio** que se llama "Coche Ford" con un saldo inicial de 0 Euros, si hacemos un nuevo movimiento en esta, no nos dejará disminuir.

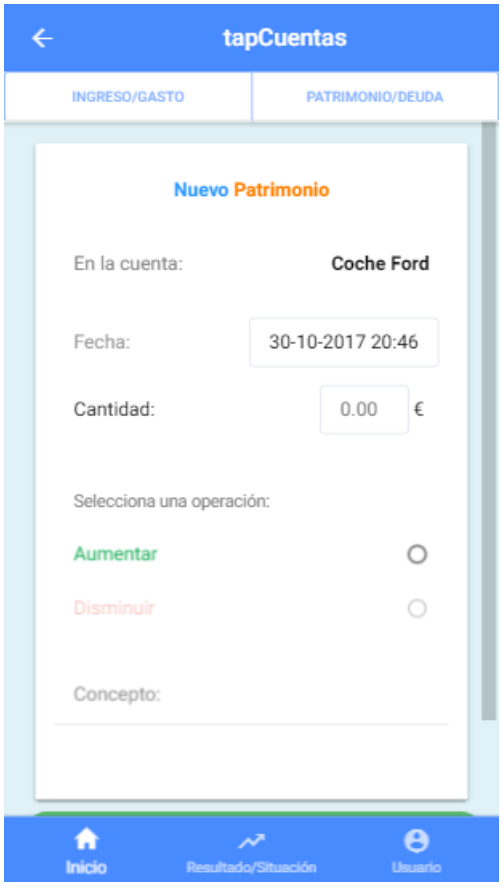

Ilustración 93 - Movimiento de tipo patrimonio en cuenta con saldo 0

Ahora bien, pongamos que tenemos una cuenta con un saldo mayor a 0 como el de la siguiente imagen.

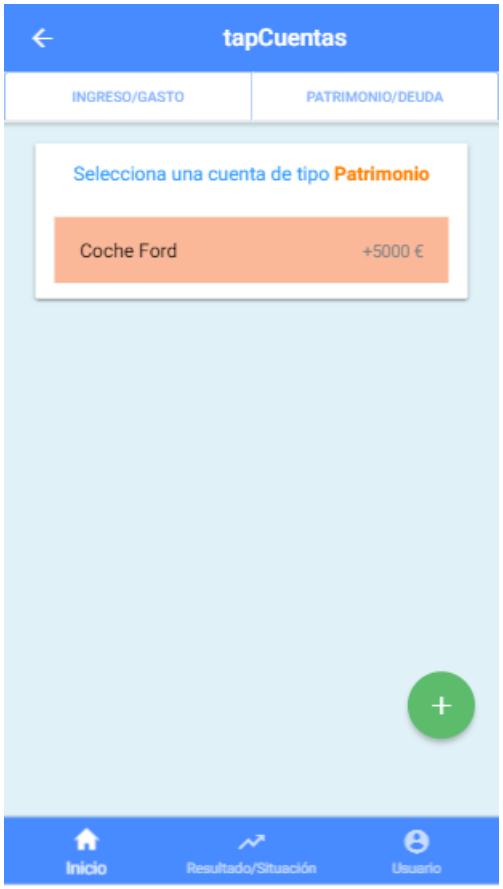

Ilustración 94 - Selección cuenta tipo Patrimonio

Podríamos seleccionar que el movimiento disminuya el saldo en la cuenta

Ilustración 95 - Selección operación disminuir en movimiento Patrimonio

Pero nunca una cantidad mayor a la que tenemos actualmente en la cuenta.

Ilustración 96 - Error en cantidad tipo Patrimonio

Este ejemplo en una cuenta de tipo *Deuda* hubiese sido muy parecido, pero en el título, el tipo de movimiento hubiera sido el correspondiente.

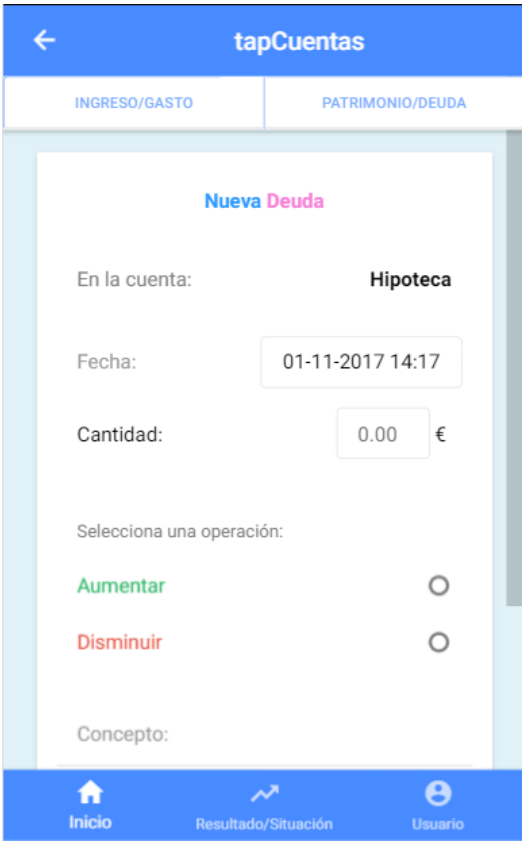

Ilustración 97 - Ejemplo creación movimiento tipo Deuda

Estos tipos de movimientos cuentan igualmente con las mismas validaciones que los anteriores, pero añadiendo de la validación de que el usuario ha seleccionado una de las dos operaciones, *aumentar o disminuir.*

Ilustración 98 - Error selección operación obligatoria

Una vez aceptamos el movimiento, salta el mensaje de aviso y volvemos al menú principal.

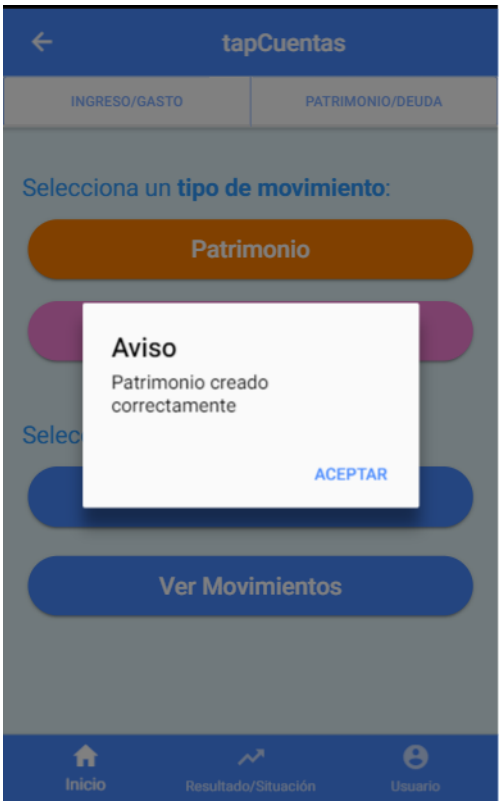

Ilustración 99 - mensaje confirmación movimiento Patrimonio

10.4 Ver Cuentas

En nuestras dos secciones también disponemos de una acción de ver nuestras cuentas.

En cada una de ellas accederemos a ver las cuentas del tipo correspondiente, en la primera, las relacionadas con **Ingreso/Gasto** y en la segunda las de **Patrimonio/Deuda** pudiendo diferenciarse en dos secciones también.

10.4.1 INGRESO/GASTO

Cuando pulsamos sobre ver las cuentas de esta sección, aparecemos por defecto en las cuentas de tipo Ingreso, si quisiéramos ver las de tipo Gasto, deberíamos pulsar sobre el segundo bloque en la barra de navegación situada bajo el título y de la misma forma, al contrario.

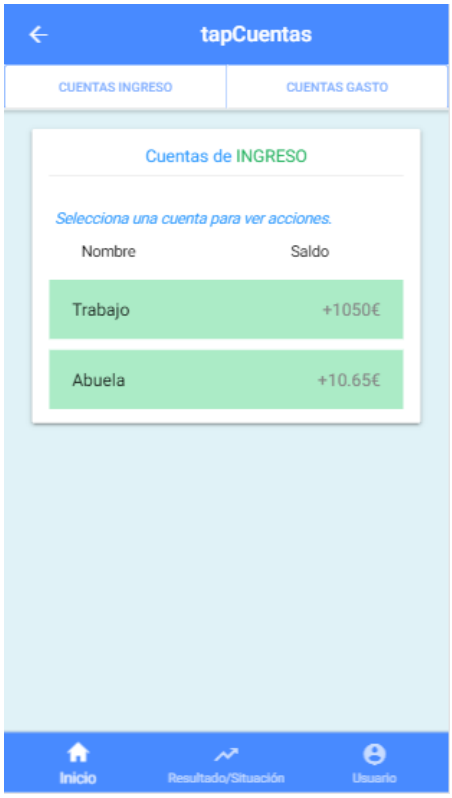

Ilustración 100 - Ver cuentas tipo Ingreso

Ilustración 101 - Ver cuentas tipo Gasto

10.4.2 PATRIMONIO/DEUDA

Si pulsamos sobre la opción *Ver Cuentas* de la parte de Patrimonio/Deuda accedemos a sus respectivas vistas.

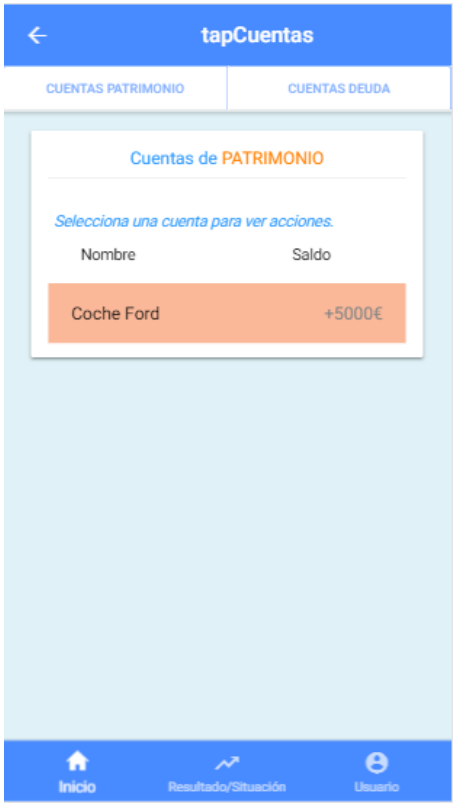

Ilustración 102 - Ver cuentas tipo Patrimonio

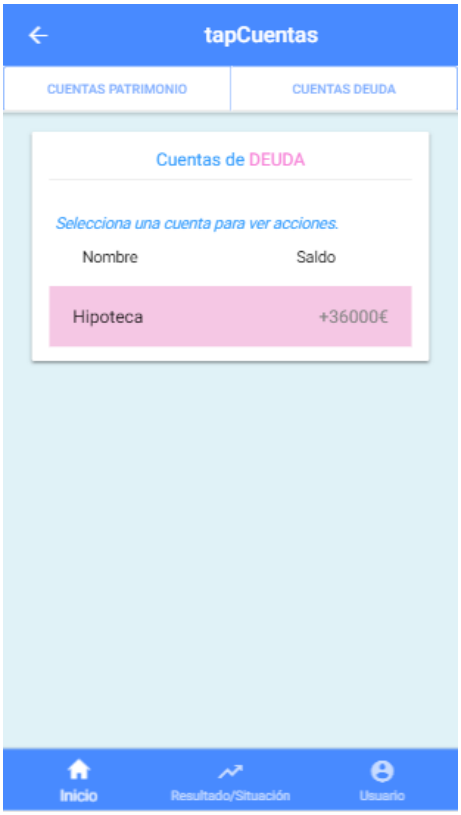

Ilustración 103 -Ver cuentas tipo Deuda

Si en alguno de los cuatro casos no existieran cuentas, la vista sería similar en cualquiera de estas.

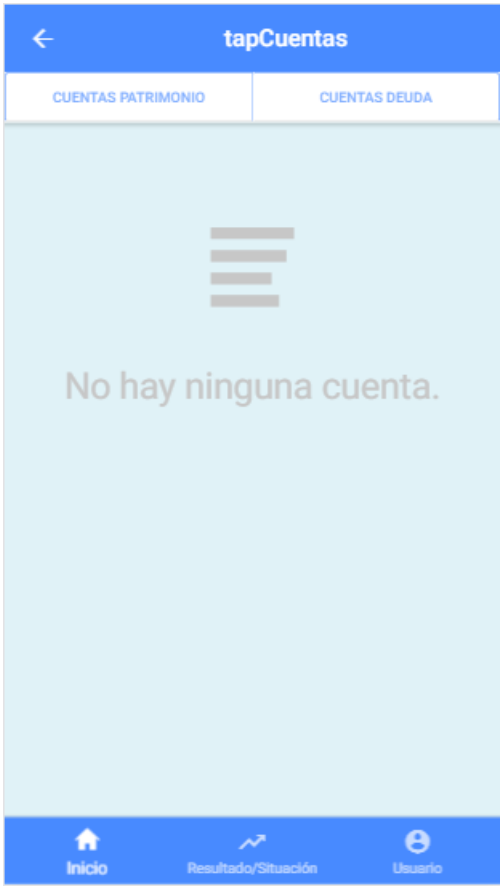

Ilustración 104 - Ver cuentas, caso sin cuentas

10.4.3 OPCIONES

Cuando accedemos a ver las cuentas de cualquier tipo, si pulsamos sobre el nombre de una cuenta, sea el tipo que sea observamos que disponemos de tres opciones:

- Ver
- Modificar
- Borrar

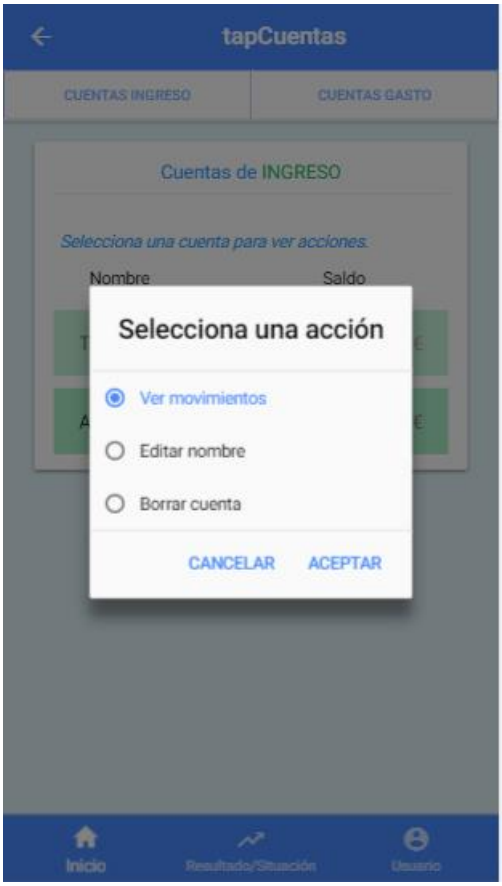

Ilustración 105 – Opciones sobre una cuenta ingreso

10.4.3.1 Ver Movimientos

Cuando entramos aquí, vemos todos los movimientos de una cuenta en concreto. Cada tipo de cuenta tiene su color correspondiente.

Ilustración 106 - Movimientos de la cuenta de ingreso 'Trabajo'

Ilustración 107 - Movimientos de la cuenta de gasto 'Mercadona'

Ilustración 108 - Movimientos de la cuenta de patrimonio 'Coche Ford'

Ilustración 109 - Movimientos de la cuenta deuda 'Hipoteca'

Si el movimiento tiene asociado algún concepto se mostrará, si no, no.

Al pulsar en un movimiento, accedemos a los detalles de este.

Ilustración 110 - Detalles movimiento de la cuenta 'Hipoteca'

A demás de ver los detalles podremos modificarlo, el proceso de modificar un movimiento lo veremos más en detalle en siguiente apartado **5.5 Ver movimientos** puesto que la pantalla de modificación es la misma.

Por ahora, centrémonos sólo en que pasaría después de modificar un movimiento desde este apartado.

Veamos un ejemplo; tenemos una cuenta 'Trabajo' de tipo Ingreso que tiene actualmente un saldo total de 1050€.

Ilustración 111 - Detalle movimientos cuenta ingreso 'Trabajo'

Modificaremos el segundo movimiento, pasando de 550€ a 500€, cuando hagamos el cambio se nos redirigirá a la pantalla de la vista de los movimientos de la cuenta 'Trabajo' con los datos **actualizados** junto a un mensaje de aviso de que todo ha ido correcto.

Ilustración 112 - Modificación segundo movimiento cuenta 'Trabajo'

Ilustración 113 - Mensaje aviso movimiento modificado correctamente

Ilustración 114 - Vista movimientos cuenta 'Trabajo' actualizada

Como se puede ver en la imagen anterior, el segundo movimiento ha pasado a 500€ y el saldo total a 1000€

Si en la modificación le diésemos al botón **cancelar** volveríamos a la vista de los *movimientos* tal y como se encontraba.

10.4.3.2 Modificar

Aquí podremos modificar el nombre de la cuenta.

Ilustración 115 - Modificación nombre cuenta

Esta pantalla tiene las mismas validaciones que la de creación de una cuenta nueva.

Ilustración 116 - Validación campo en blanco

Si modificamos el nombre correctamente, nos mostrará un mensaje de aviso y nos redirigirá a la pantalla anterior.

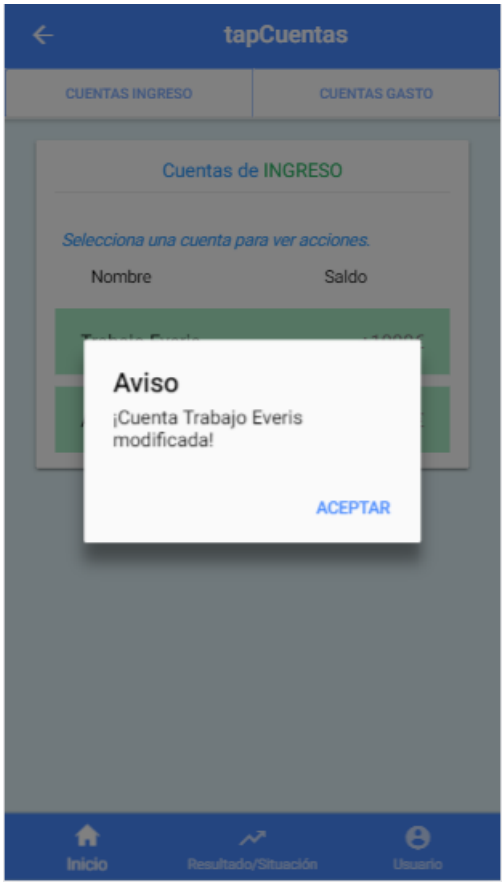

Ilustración 117 - Nombre cuenta modificado correctamente

Ilustración 118 - Datos de la cuenta con nuevo nombre

Si le diéramos al botón **cancelar** volveríamos a la vista de los *movimientos.*

10.4.3.3 Borrar

Si pulsamos este botón, se mostrará un mensaje de confirmación (como en toda operación delicada) y si se confirma se borrará, tanto la cuenta seleccionada como sus movimientos asociados.

Veamos un ejemplo, vamos a borra la cuenta Ingreso 'Abuela'.

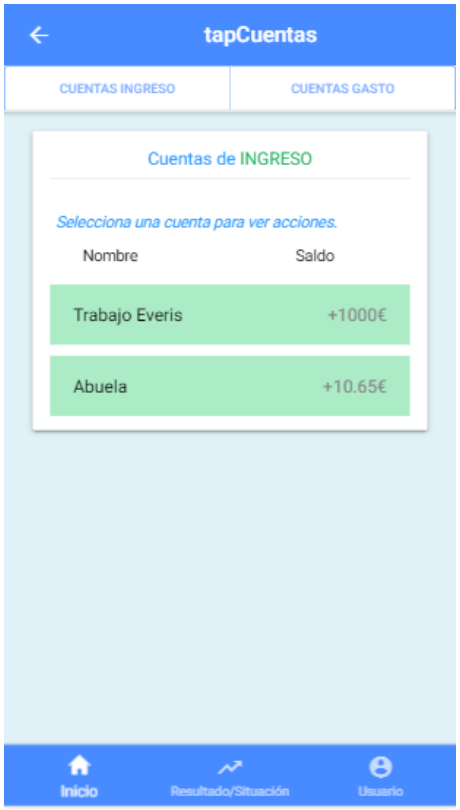

Ilustración 119 - Pantalla Cuentas antes de borrado

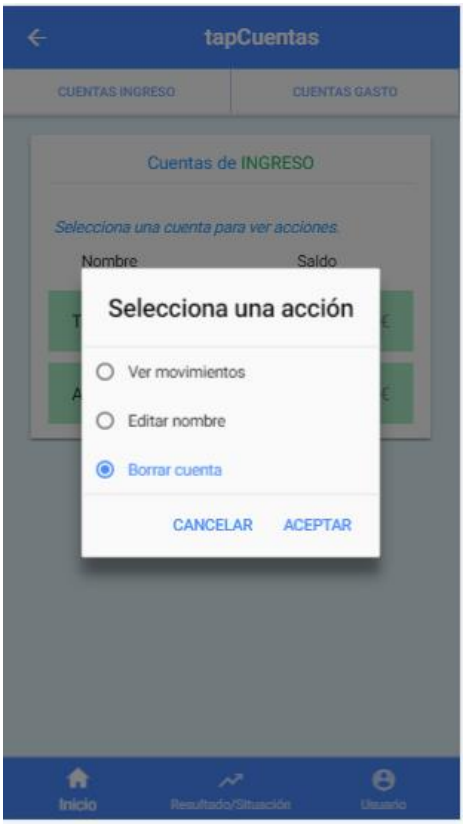

Ilustración 120 - Selección 'Borrar cuenta'

Ilustración 121 - Confirmación borrado cuenta correctamente

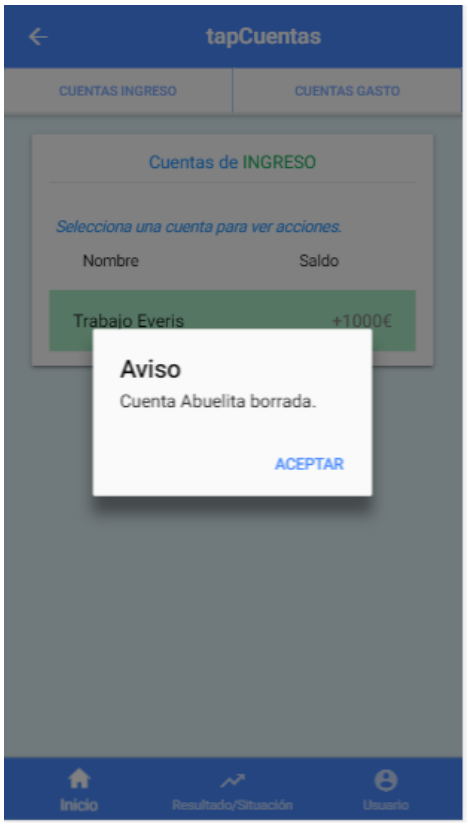

Ilustración 122 - Aviso movimiento borrado

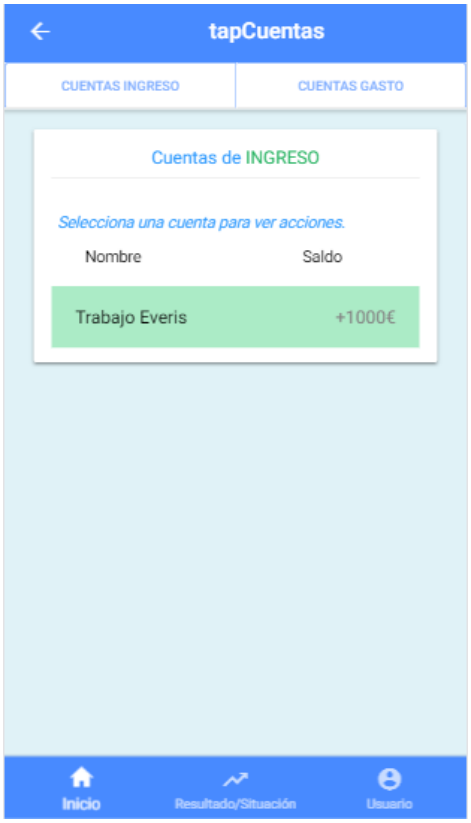

Ilustración 123 – Vista cuentas después de borrado

Como observamos, la cuenta se ha borrado correctamente, si le diésemos a la opción 'No', no haría nada.

10.5 Ver Movimientos

Cuando pulsamos la acción de 'Ver Movimientos' accedemos a una lista de movimientos ordenados por fecha.

Si accedemos a ver movimientos desde el bloque de **INGRESO/GASTO**, veremos los movimientos de ingreso y gasto.

Ilustración 124 - Ver movimientos de Ingresos y Gastos

Tenemos una opción de **filtrar** los movimientos por tipo en el caso de que se quisieran ver únicamente de un determinado tipo o de ambos.

Ilustración 125 - Opción filtro movimientos Ingreso

Ilustración 126 - Filtro movimientos ingreso

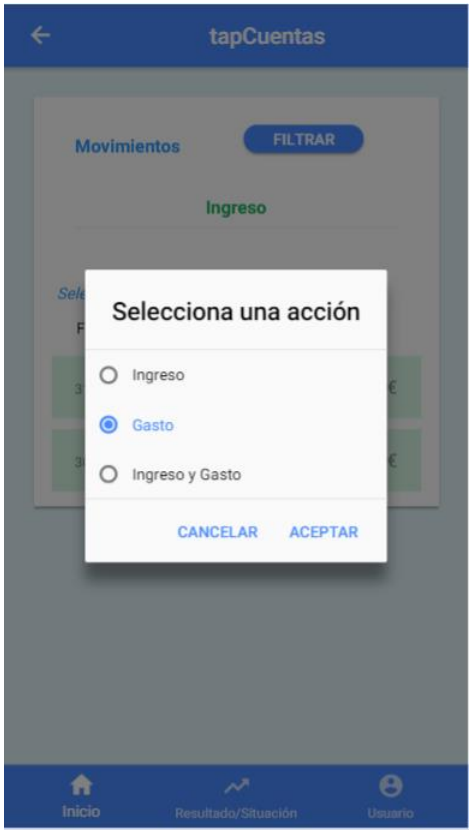

Ilustración 127 - Opción filtro movimientos Gastos

Ilustración 128 - Filtro movimientos Gastos

Si por el contrario accedemos a 'Ver movimientos' desde **PATRIMONIO/DEUDA**, tendremos los movimientos de las cuentas de estos tipos.

Ilustración 129 - Ver movimientos Patrimonio/Deuda

Contando con los correspondientes filtros.

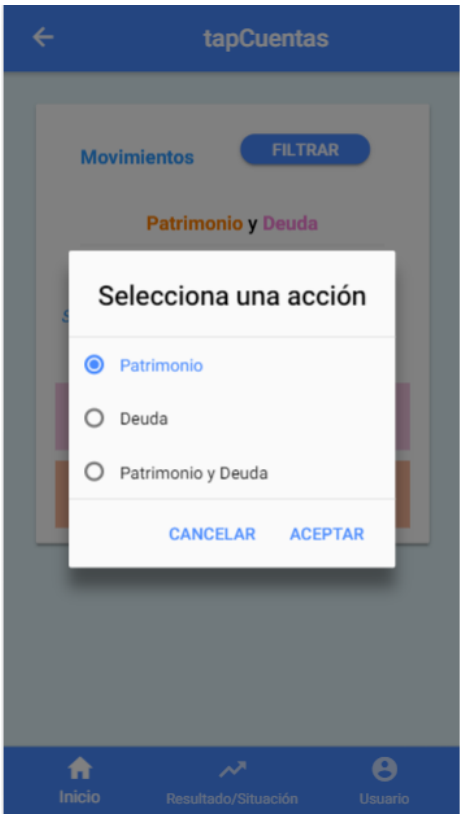

Ilustración 130 - Opción filtro movimientos

Ilustración 131 - Filtro movimientos Patrimonio

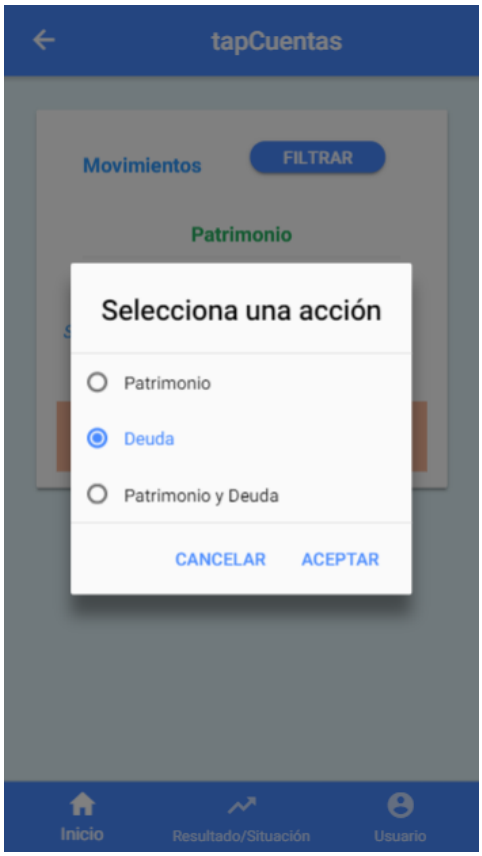

Ilustración 132 - Opción filtro movimientos Deuda

Ilustración 133 - Filtro movimientos Deuda

Si no hubiera ningún movimiento de ninguno de los dos tipos principales, la pantalla mostrada seria la siguiente:

Ilustración 134 - Pantalla 'Ver Movimientos' sin movimientos

Al pulsar en cualquiera de los movimientos, accedemos a los detalles de este.

Ilustración 135 - Pantalla detalles de movimiento tipo Gasto

La pantalla de detalle de un movimiento de tipo Patrimonio o tipo Deuda corresponde al mismo estilo que el de creación de estos.

Ilustración 136 - Pantalla detalles de movimiento tipo Patrimonio

Las pantallas de detalle de movimiento también contienen sus respectivas validaciones.

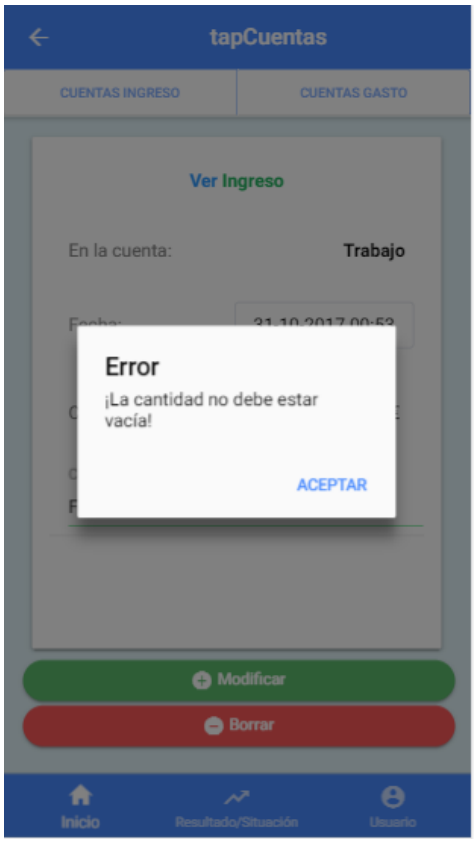

Ilustración 137 - Validación campo saldo vacío mayor a 0

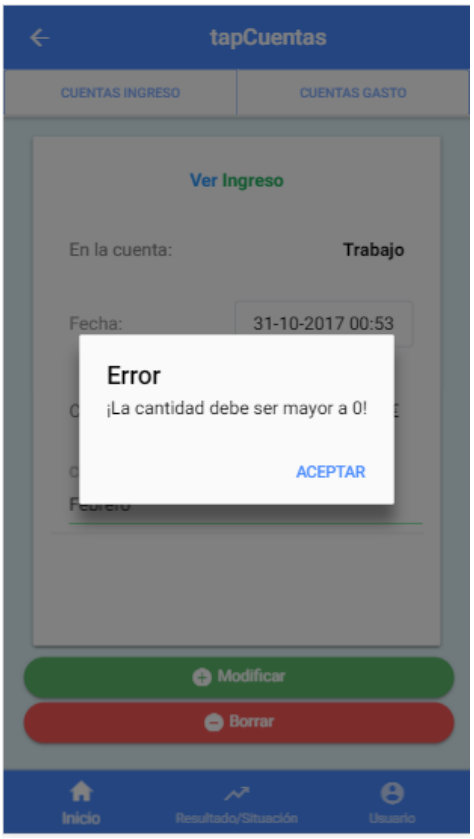

Ilustración 138 - Validación cantidad

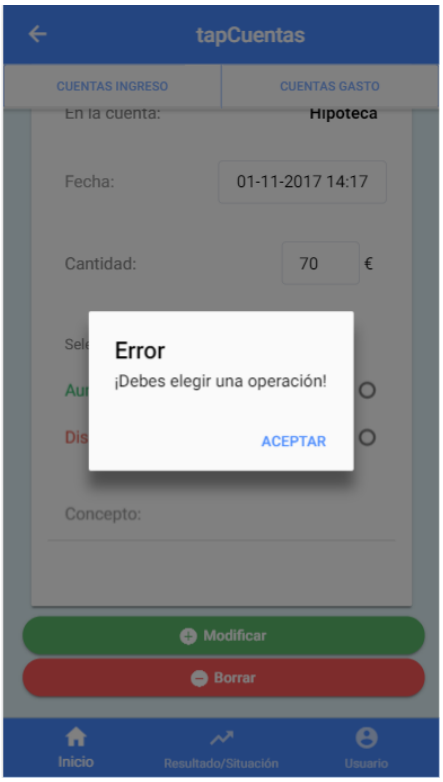

Ilustración 139 - Validación selección operación

Ilustración 140 - Validación cantidad en disminución

Si modificamos correctamente un movimiento, volveremos a la pantalla del listado de movimientos correspondiente.

Ilustración 141 - Movimiento modificado correctamente

Si en el detalle le diésemos a **cancelar** simplemente volveríamos al listado de movimientos.

10.6 Sección Resultados

A esta sección accedemos desde el menú de navegación inferior, en el icono de la flecha con título '**Resultados/Situación'.**

La primera pantalla que vemos es la de **Resultados**, este es el aparado relacionado con los **ingresos** y **gastos**, en ella podemos ver nuestros resultados financieros actuales, indicándonos:

- Ingresos totales.
- Gastos totales.
- Si nuestro resultado total es de AHORRO**,** en el caso de que nuestros ingresos sean mayores que nuestros gastos, o de **DÉFICIT** en el caso contrario.
- Un listado de las cuentas de Ingreso y Gasto con su saldo total.

Si tenemos más Ingresos que Gastos nuestro resultado es AHORRO.

Ilustración 142 - Pantalla resultados situación ahorro

Si por el contrario tuviesemos mas Gastos que Ingresos, nuestro resultado sería DEFICIT.

Ilustración 143 - Pantalla resultados situación déficit

Además, como detalle visual, para que el usuario pueda ver que se muestran lso dos tipos de cuentas, si en uno de los resumen de cuentas de algun tipo existieran más de dos cuentas, se crearia una lista desplegable, veamos un ejemplo.

Tenemos tres cuentas de tipo gasto, en principio sólo se muestran dos, pero con un pequeño botón den forma de flecha que nos indica que existe un desplegable.

Ilustración 144 - Lista cuentas gastos sin desplegar

Si le pulsamos, accedemos a la lista entera

Ilustración 145 - Lista cuentas gastos desplegada

Si le volviésemos a pulsar a la flecha invertida, esta lista de volvería comprimir.

Si no existieran movimientos de un tipo de cuentas se mostraría de la siguiente manera:

Ilustración 146 - Resultados sin cuentas tipo Gasto

10.7 Sección Situación

Si desde la pantalla de **Resultados** pulsamos sobre el segundo bloque que aparece en la parte superior de la pantalla con título '**SITUACIÓN** accederemos a esta sección, en ella visualizamos el apartado de **Situación**, es muy parecido al de '**Resultados'** (por lo que algunos detalles no los repetiremos) pero este es el aparado relacionado con el **patrimonio** y **deudas**, en él podemos ver nuestra situación financiera actual, indicándonos:

- Patrimonio total.
- Deudas totales.
- Si en nuestra situación disponemos de **CAPITAL,** en el caso de que nuestro patrimonio sea mayor que nuestras deudas, o de **QUIEBRA** en el caso contrario.

Un listado de las cuentas de Patrimonio y Deudas con su saldo total.

Actualmente tenemos más saldo en cuentas de Patrimonio que en deuda. Por lo que nos indica que nuestro resultado es de **CAPITAL.**

Ilustración 147 - Pantalla situación caso capital

Si por el contrario, tuviesemos mas saldo en las cuentas de deudas, nuestra situación seía de **QUIEBRA.**

Ilustración 148 - Pantalla situación caso quiebra

10.8 Sección Usuario

Por último, tenemos la sección del usuario, a la cual accedemos desde el último icono de la barra inferior de navegación que tiene de icono de un usuario y de título 'Usuario'. Aquí tendremos disponibles las opciones relacionadas con el usuario.

- Modificar usuario
- Cerrar sesión
- Borrar cuenta

Ilustración 149 -Pantalla principal usuario

10.8.1 MODIFICAR USUARIO

En esta primera opción podremos cambiar los datos del usuario, tanto el email como la contraseña.

Ilustración 150 - Pantalla modificar usuario

Para modificar el email, es necesario rellenar el campo con un formato de email válido.

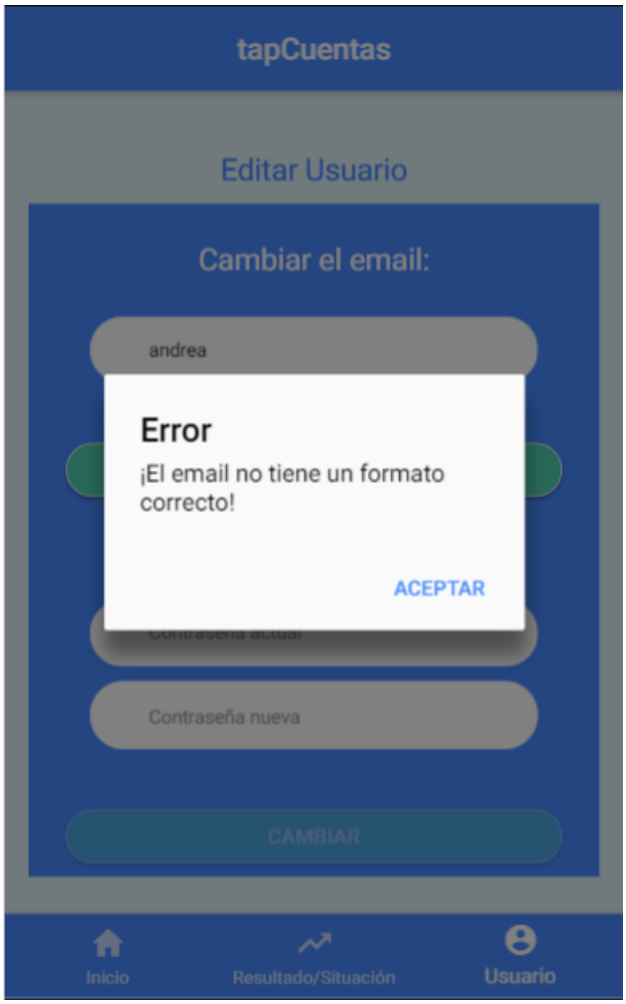

Ilustración 151 - Error formato email

Si modificamos el email correctamente, nos avisará con un mensaje.

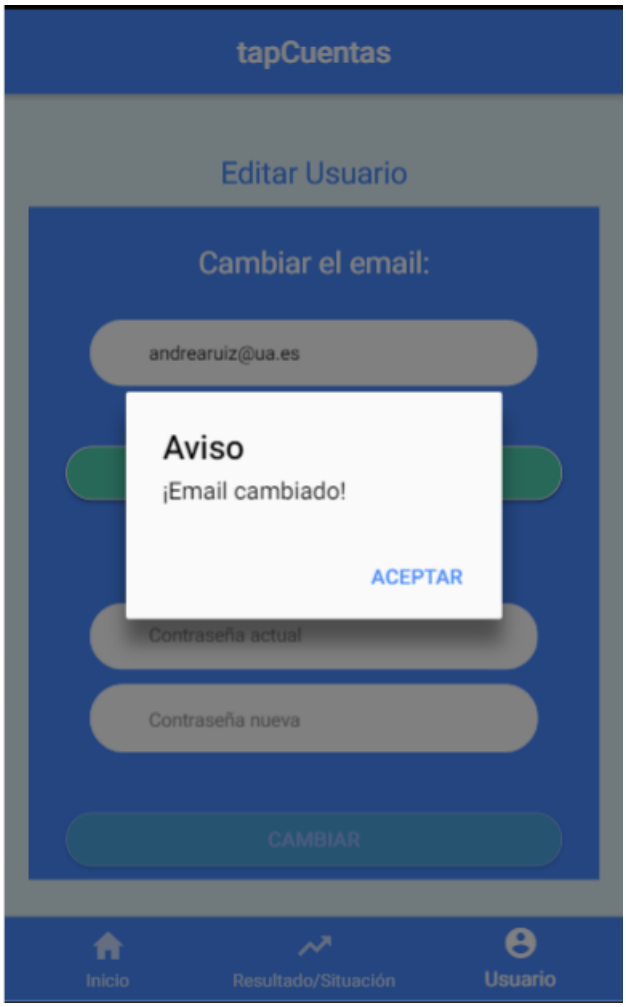

Ilustración 152 - Email modificado correctamente

En el caso de la contraseña es necesario escribir la contraseña anterior como medida de seguridad.

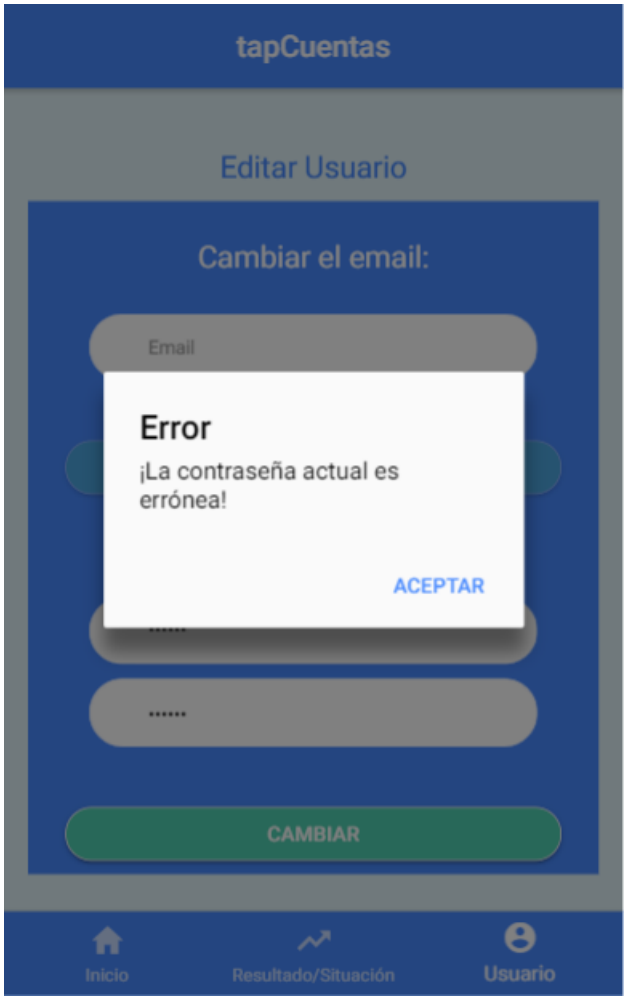

Ilustración 153 - Confirmación contraseña actual errónea

Si la modificáramos correctamente, nos saldría un mensaje de aviso.

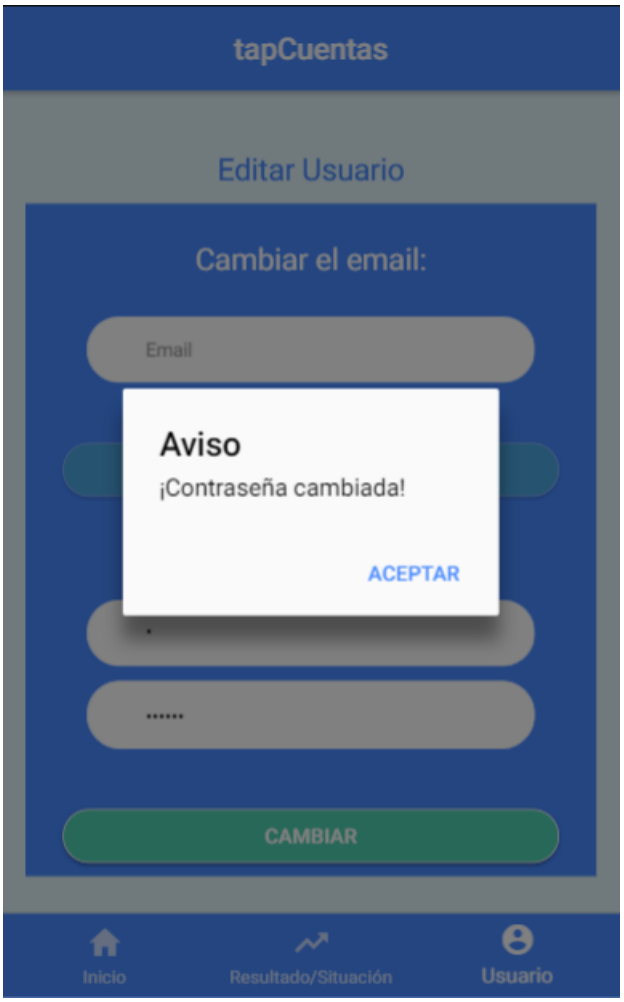

Ilustración 154 - Contraseña modificada correctamente

Para poder pulsar sobre el botón de realizar el cambio es necesario llenar los datos, de lo contrario permanecerá inhabilitado.

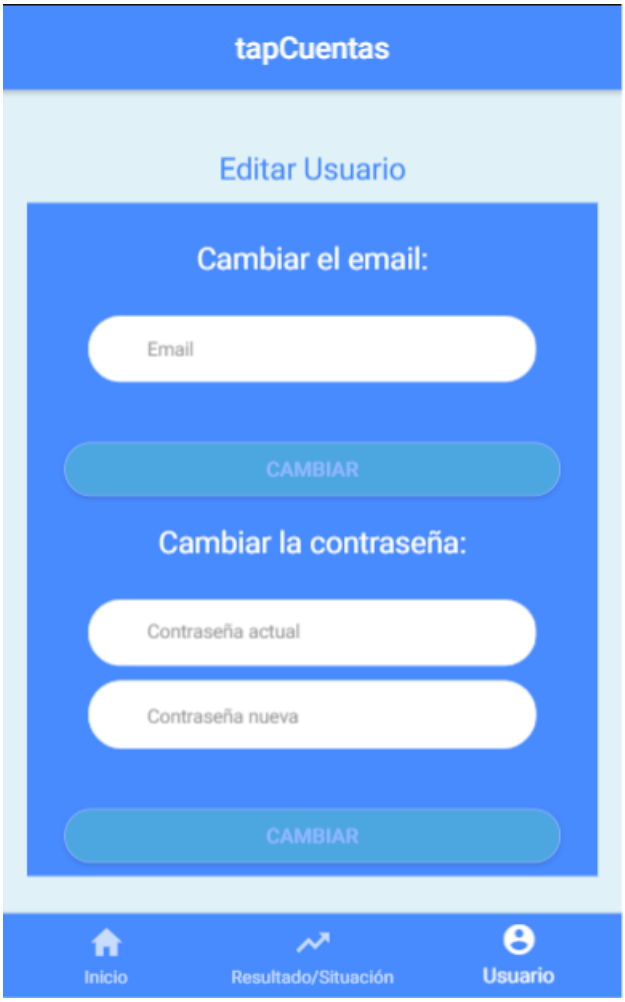

Ilustración 155 - Campos vacíos pantalla edición usuario

10.8.2 CERRAR SESIÓN

Este aparato simplemente saca al usuario de la aplicación, volviendo a la pantalla de *login* y borrando la variable global de usuario conectado almacenada.
Muestra un mensaje de confirmación antes de salir como medida de prevención a errores.

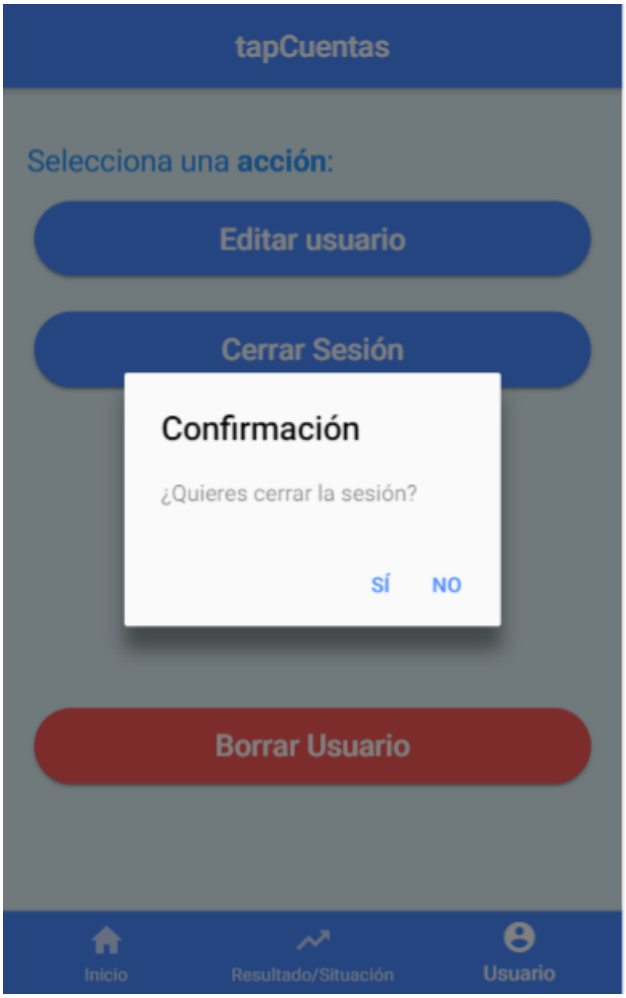

Ilustración 156 - Mensaje confirmación cerrar sesión

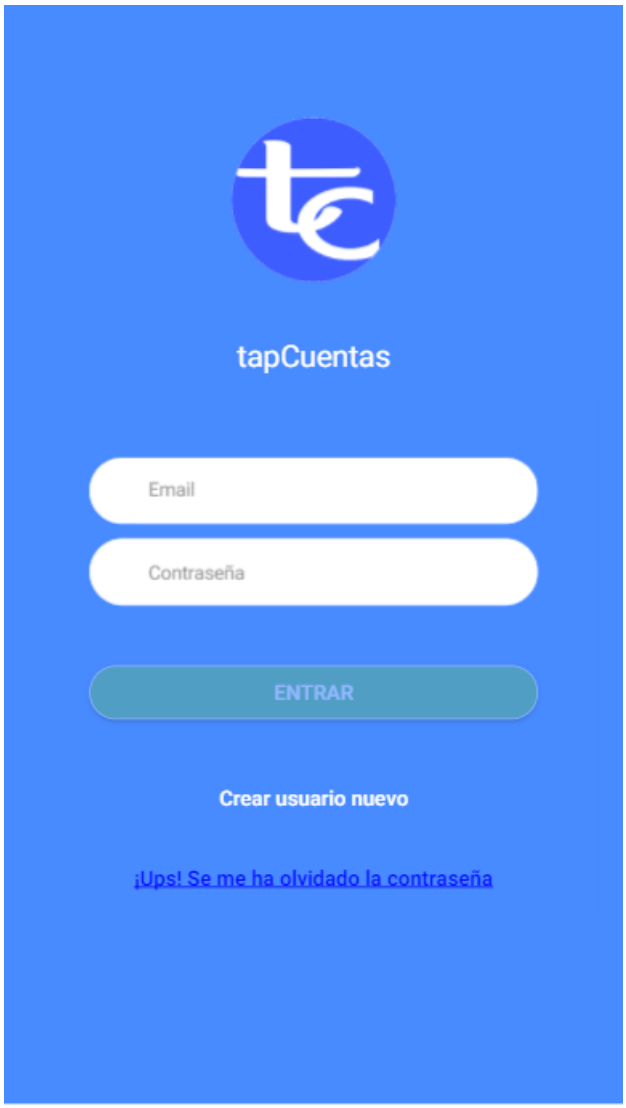

Ilustración 157 - Pantalla login después de cerrar sesión

10.8.3 BORRAR CUENTA

Esta opción, borra la cuenta del usuario junto a sus cuentas y movimientos de la base de datos y sale de la aplicación.

También cuenta con un mensaje de confirmación para evitar posibles erres al pulsar.

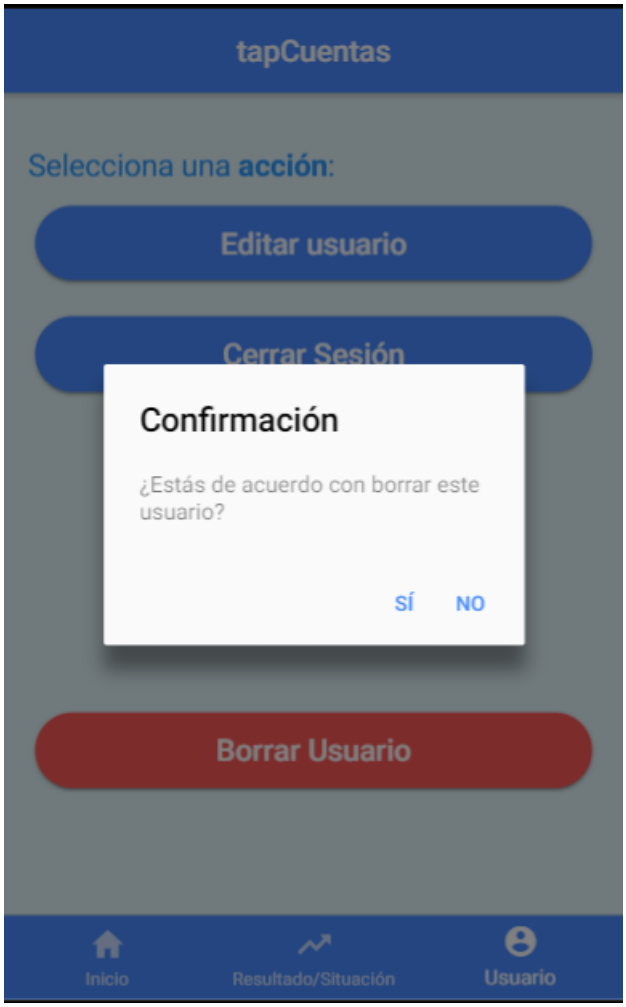

Ilustración 158 - Mensaje confirmación borrar usuario

11. Referencias

Aquí exponemos algunas de las referencias que nos han ayudado o hemos usado durante el proyecto.

Cli.angular.io. (2017). Angular CLI. Enlace externo: https://cli.angular.io/

Es.wikipedia.org. (2017). Angular (framework). Enlace externo: https://es.wikipedia.org/wiki/Angular_%28framework%29

Es.wikipedia.org. (2017). Objeto de acceso a datos. Enlace externo: https://es.wikipedia.org/wiki/Objeto_de_acceso_a_datos

Es.wikipedia.org. (2017). Trello. Enlace externo: https://es.wikipedia.org/wiki/Trello Es.wikipedia.org. (2017). TypeScript. Enlace externo: https://es.wikipedia.org/wiki/TypeScript

Hub.docker.com. (2017). Enlace externo: https://hub.docker.com/_/mongo/

Ionic Framework. (2017). Ionic Framework. Enlace externo: https://ionicframework.com/docs/

Kanbantool.com. (2017). Metodología Kanban | Kanban Tool. Enlace externo: https://kanbantool.com/es/metodologia-kanban

Tools, S., Editor, S., Codegen, S., UI, S., Inspector, S., Tools, C., Integrations, O., Discussion, R., Forum, O. and Champions, S. (2017). OpenAPI Specification | Swagger. Swagger. Enlace externo: https://swagger.io/specification/

12. Enlaces externos

Aunque ya se ha comentado durante la memoria, en este apartado expondré los enlaces a mi proyecto que con gusto dejare abiertos para su observación si se desea.

Código tapCuentas en GitHub: https://hub.docker.com/u/alirium/
Docker del proyecto: https://github.com/alirium/TFG---tapCuentas